ISO TC 184/SC4/ WG12 Nssss

Date: 2000/09/22

Supersedes ISO TC 184/SC4/ __ N ___

ISO/WD 10303-5s

Product data representation and exchange: Integrated resource: Mesh-based topology

ABSTRACT:

This provides an initial draft of the mesh-based topology resource.

KEYWORDS: Mesh, Topology

COMMENTS TO READER:

This document presents a harmonization of the **topological_definition_and_mesh**, **unstructured_mesh** and **structured_mesh** schemas from N605, and the **mesh_topology** and **data_array** schemas from the putative Part 5w. It is a revised version of the 2000/08/17 document. The formal modeling uses EXPRESS, Amendment 1.

Project Leader: Ray Cosner
Address: Boeing, Phantom Works
PO BOx 516,
M/S S106–7126
St. Louis, MO 63166

Telephone: +1 (314) 233–6481
Telefacsimile: +1 (314) 777–1328
Electronic mail: raymond.r.cosner@boeing.com

Project Editor: Peter Wilson
Address: Boeing Commercial Airplane
PO Box 3707, M/S 6H-AF
Seattle, WA 98124–2207

Telephone: +1 (425) 237–3506
Telefacsimile: +1 (425) 327–3428
Electronic mail: peter.r.wilson@boeing.com

International Organization for Standardization
Case Postale 56 • CH–2111 Genève 20 • Switzerland

Contents

Figures

Tables

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75% of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

International Standard ISO 10303–5s was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data*.

This International Standard is organized as a series of parts, each published separately. The parts of ISO 10303 fall into one of the following series: description methods, integrated resources, application interpreted constructs, application protocols, abstract test suites, implementation methods, and conformance methods. The series are described in ISO 10301-1.

A complete list of parts of ISO 10303 is available from the Internet:
<http://www.nist.gov/sc4/editing/step/titles/>

This part of ISO 10303 is a member of the integrated resource series.

Annexes A and B are a normative part of this International Standard. Annexes C and D are for information only.

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

Major subdivisions of this International Standard are:

— **mesh_topology_schema**;

— **data_array_schema**.

The relationships of the schemas in this part of ISO 10303 to other schemas that define the integrated resources of this International Standard are illustrated in Figure 1 using the EXPRESS-G notation. EXPRESS-G is defined in annex D of ISO 10303-11. The schemas identified in the bold boxes are specified in this part of ISO 10303. The **measure_schema** and **support_resource_schema** are specified in part 41 of ISO 10303. The **topology_schema**, **mathematical_functions_schema**, and **structural_response_representation_schema** are specified in parts 42, 50, and 104 of ISO 10303, respectively. The schemas illustrated in Figure 1 are components of the integrated resources.

There are many applications that have to deal with massive amounts of data, which is normally numerical in nature. The quantity of data may be measured in gigabytes and in some cases terabytes. Examples include computational fluid dynamics, dynamic simulation of vehicle behaviour, and experimental data of many kinds ranging from high energy physics to global weather measurements.

A major concern in dealing with such data is to optimise the data representation and structure with respect to data transmission and storage. As part of the optimisation, the data tends to be maintained in large arrays where any particular data element can be referenced by a simple index into the array. When the data is part of a computer simulation the data is usually associated with a mesh of some kind — either structured or unstructured. The data may be bound to the vertices of the mesh or to the cells of the mesh. In any case, it is also possible to represent the simpler kinds of meshes by an indexing scheme. Within this part illustrative examples have been principally taken from the field of computational fluid dynamics.

This part of ISO 10303 provides general, application independent, means of representing indexible data and meshes.

In this International Standard the same English language words may be used to refer to an object in the real world or to a concept, and as the name of an EXPRESS data type that represents this object or concept. The following typographical convention is used to distinguish between these. If a word or phrase occurs in the same typeface as narrative text, the referent is the object or concept. If the word or phrase occurs in a bold typeface, the referent is the

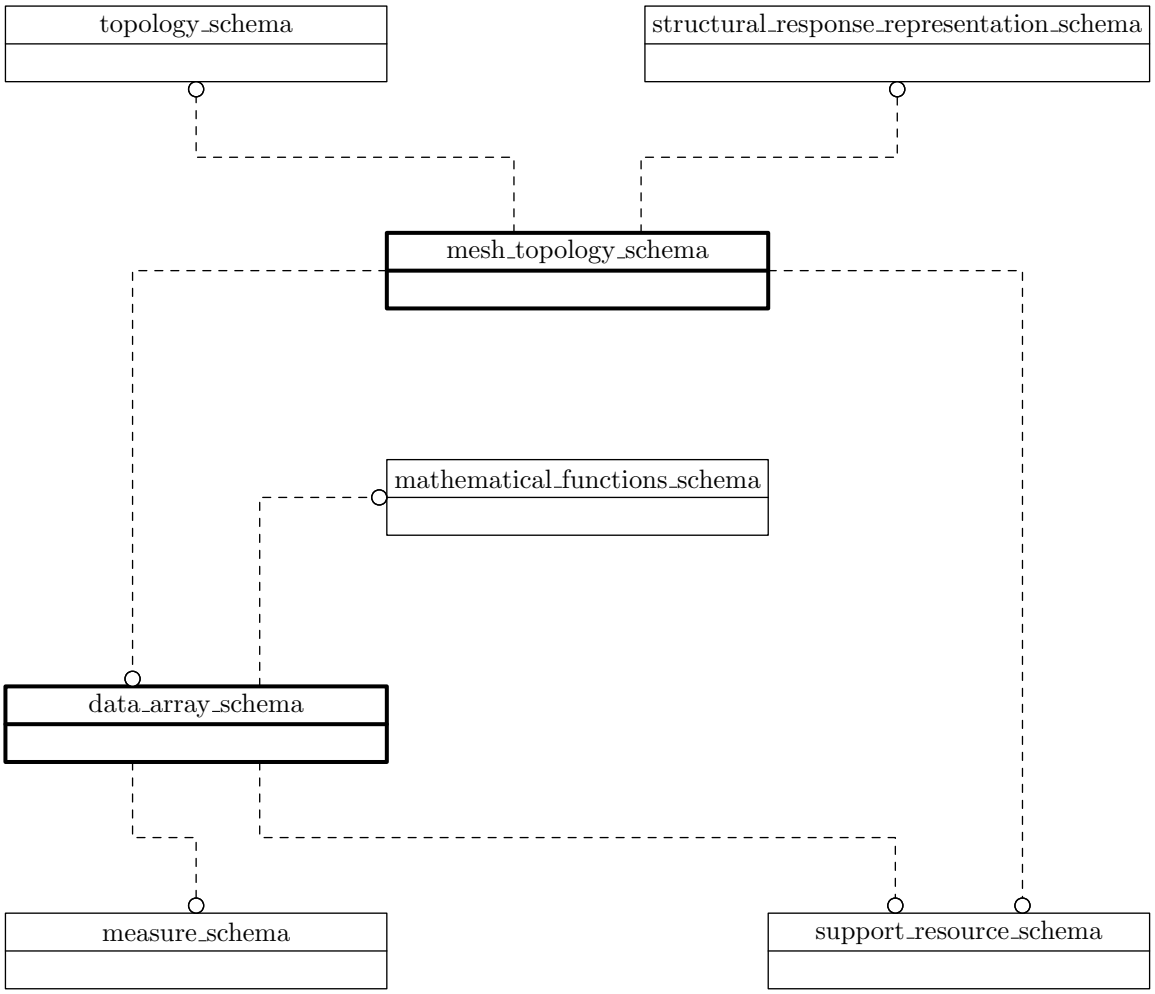


Figure 1 – Schema relationships

EXPRESS data type. Names of EXPRESS schemas also occur in a bold typeface.

The name of an EXPRESS data type may be used to refer to the data type itself, or to an instance of the data type. The disctinction between these uses is normally clear from the context. If there is a likelihood of ambiguity, the phrase 'entity data type' or 'instance(s) of' is included in the text.

Quotation marks " " are used to denote text that is copied from another document. Inverted commas '' are used to denote particular string values.

Several components of this part of ISO 10303 are available in electronic form. This access is provided through the specification of Universal Resource Locators (URLs) that identify the location of these files on the Internet. If there is difficulty accessing these sites contact the ISO Central Secretariat or the ISO TC184/SC4 Secretariat directly at: `sc4@cme.nist.gov`.

Industrial automation systems and integration — Product data representation and exchange — Part 5s : Integrated resource: Mesh-based topology

1 Scope

The following are within the scope of this part of ISO 10303:

— Mesh-based topologies;

— Data arrays.

The following are outside the scope of this part of ISO 10303:

— Applications of mesh topologies;

— Applications of data arrays.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this international standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this international standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO 10303-1:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles.*

ISO 10303-11:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.*

ISO/TR 10303-12:1997, *Industrial automation systems and integration — Product data representation and exchange — Part 12: Description methods: The EXPRESS-I language reference manual.*

ISO 10303-41:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 41: Integrated resource: Fundamentals of product description and support.*

ISO 10303-42:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated resource: Geometric and topological representation.*

ISO 10303-50:2000[1)], *Industrial automation systems and integration — Product data representation and exchange — Part 50: Integrated resource: Mathematical constructs.*

ISO 10303-104:1999[1)], *Industrial automation systems and integration — Product data representation and exchange — Part 104: Integrated application resource: Finite element analysis.*

ISO/IEC 8824-1:1995, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

3 Terms, definitions, abbreviations, and symbols

3.1 Terms defined in ISO 10303-1

— application protocol (AP)

3.2 Terms defined in ISO 10303-12

— object base

3.3 Terms defined in ISO 10303-42

— graph

3.4 Other definitions

3.4.1
cell
a connected topological region of dimensionality one or higher that is a part of, or the whole of, the domain of a mesh.

3.4.2
mesh
an arrangement of topological regions of zero or higher dimensionality with connectivity between the topological regions defined by the possession of common faces, edges or bounds.

3.4.3

[1)] To be published.

Table 1 – Symbols for dimensional units

Symbol	Description
M	mass unit
L	length unit
T	time unit
Θ	temperature unit
α	angle unit

Table 2 – Symbols for coordinate systems

Symbol	Description
x, y, z	coordinates in a Cartesian system
r, θ, z	coordinates in a Cylindrical system
r, θ, ϕ	coordinates in a Spherical system
ξ, η, ζ	coordinates in an auxiliary system

node
a location in a cell that is not a (bounding) vertex.

3.4.4
topological region
a point set with a single topological dimension.

3.4.5
vertex
a bound of a cell.

3.5 Abbreviations

CFD computational fluid dynamics

URL Universal Resource Locator

3.6 Symbols

Symbols for dimensional units are given in Table 1.

EXAMPLE 1 A length has dimensions **L**, an area has dimensions $\mathbf{L}^2$, and a velocity has dimensions $\mathbf{L/T}$ (alternatively written as $\mathbf{LT}^{-1}$).

Symbols for coordinate systems are given in Table 2.

Associated with the coordinate systems are unit vectors, the symbols for which are given in Table 3.

Table 3 – Symbols for unit vectors

Symbol	**Direction**	**Symbol**	**Direction**	**Symbol**	**Direction**
$\hat{e}_x$	x-direction	$\hat{e}_r$	r-direction	$\hat{e}_\xi$	ξ-direction
$\hat{e}_y$	y-direction	$\hat{e}_\theta$	θ-direction	$\hat{e}_\eta$	η-direction
$\hat{e}_z$	z-direction	$\hat{e}_\phi$	ϕ-direction	$\hat{e}_\zeta$	ζ-direction

4 mesh_topology_schema

The following EXPRESS declaration begins the **mesh_topology_schema** and identifies the necessary external references.

EXPRESS specification:

```
*)
{iso standard 10303 part (11) version (4)}
SCHEMA mesh_topology_schema;
  REFERENCE FROM data_array_schema
    (data_array);
  REFERENCE FROM topology_schema
    (topological_representation_item,
     vertex);
  REFERENCE FROM structural_response_representation_schema
    (element_order);
  REFERENCE FROM support_resource_schema
    (label
     text);
(*
```

NOTE The schemas referenced above can be found in the following parts of ISO 10303:

data_array_schema	Clause 5 of this part of ISO 10303
topology_schema	ISO 10303-42
structural_response_representation_schema	ISO 10303-104
support_resource_schema	ISO 10303-41

4.1 Introduction

This schema defines and describes the structure types for describing mesh topologies.

4.2 Fundamental concepts and assumptions

A mesh is defined by its vertices and the connections between the vertices. A mesh is a connected graph.

4.2.1 Regular mesh

In a regular mesh, the volume is the ensemble of cells.

In a 3–D rectangular grid a cell is the hexahedron region defined by eight vertices forming the corners of the hexahedron. Each cell is bounded by six faces, where each face is the quadrilateral made up of four vertices. An edge links two corner vertices; a face is bounded by four edges.

In a 2–D rectangular grid a cell is the quadrilateral region defined by four vertices forming the corners of the quadrilateral. Each cell is bounded by four sides, where a side is the line bounded

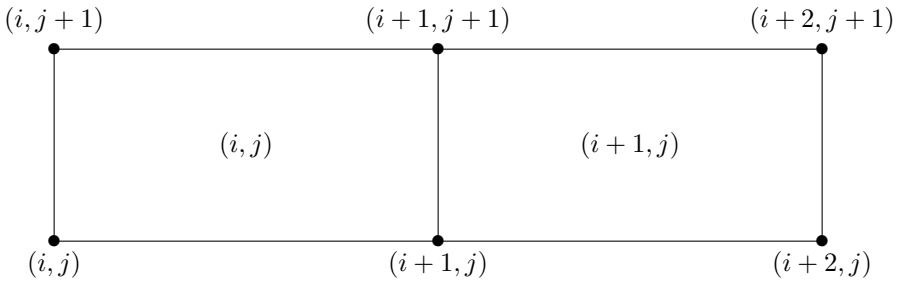


Figure 2 – Example convention for a 2-D cell center

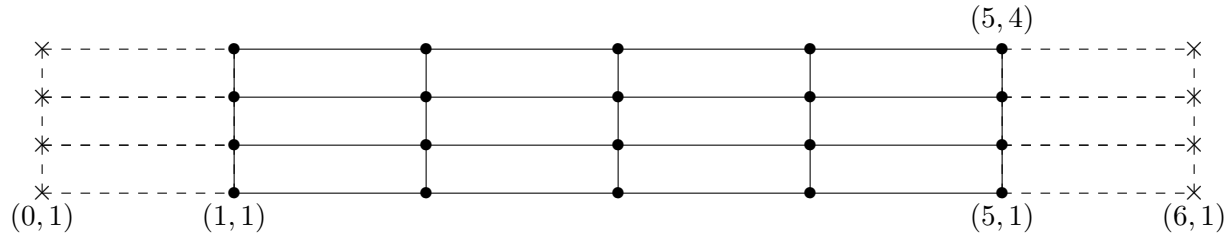


Figure 3 – Example mesh with rind vertices

by two vertices. A 'side' may be referred to as a 'face' or an 'edge'.

In a 1–D mesh a cell is the line bounded by two vertices.

Indices describing a regular mesh are ordered: for 3–D (i, j, k); (i, j) is used for 2–D; and (i) for 1–D.

Cell centers, face centers, and edge centers are indexed by the minimum of the connecting vertices.

EXAMPLE 1 For example a 2-D cell center (or face center on a 3-D mesh) would have the conventions shown in Figure 2.

In addition, the default beginning vertex for a regular mesh is $(1, 1, 1)$; this means the default beginning cell center of a regular mesh is also $(1, 1, 1)$.

There may be locations the mesh itself. These are referred to as 'rind' or ghost points and may be associated with fictitious vertices or cell centers. They are distinguished from the vertices and cells making up the mesh (including its boundary vertices), which are referred to as 'core' points.

EXAMPLE 2 Figure 3 shows a 2–D mesh with a single row of 'rind' vertices at the minimum and maximum i-faces. The mesh size (i.e. the number of 'core' vertices in each direction) is 5×4. 'Core' vertices are designated by '•', and 'rind' vertices by '×'. Default indexing is also shown for the vertices.

For a mesh, the minimum faces in each coordinate direction are denoted i-min, j-min and k-min; the maximum faces are denoted i-max, j-max and k-max. These are the minimum and

maximum 'core' faces.

EXAMPLE 3 $i-$min is the face or grid plane whose core vertices have minimum i index (which if using default indexing is 1).

4.2.2 Irregular mesh

An irregular mesh is composed of vertices and cells, where the shape of the cells is not restricted to be uniform throughout the mesh. Cells have vertices at their corners and may also have vertices (nodes) on the edges, faces, and interior of the cell.

Each cell in an irregular mesh has at least one vertex in common with at least one other cell in the mesh. The connectivity and adjacency of the cells may be determined from the common vertices.

A cell is represented in terms of its shape and an ordered list of its vertices. The vertices are implied rather than being explicitly represented. Essentially all the vertices in a mesh can be mapped to a sequential list, and reference to a vertex is then equivalent to specifying the particular position in the list.

4.3 mesh_topology_schema type definitions

4.3.1 cell_shape

An identifier of an unstructured mesh cell shape.

EXPRESS specification:

```
*)
TYPE cell_shape = EXTENSIBLE SELECT OF
    (cell_shape_0D,
     cell_shape_1D,
     cell_shape_2D,
     cell_shape_3D);
END_TYPE;
(*
```

4.3.2 cell_shape_0D

An identifier of a topologically 0–D unstructured mesh cell shape.

EXPRESS specification:

```
*)
```



```
TYPE cell_shape_0D = EXTENSIBLE ENUMERATION OF
    (single);
END_TYPE;
(*
```

Enumerated item definitions:

single: singleton vertex.

4.3.3 cell_shape_1D

An identifier of a topologically 1–D unstructured mesh cell shape.

EXPRESS specification:

```
*)
TYPE cell_shape_1D = EXTENSIBLE ENUMERATION OF
    (bar);
END_TYPE;
(*
```

Enumerated item definitions:

bar: a topological line requiring a minimum of 2 vertices.

4.3.4 cell_shape_2D

An identifier of a topologically 2–D unstructured mesh cell shape.

EXPRESS specification:

```
*)
TYPE cell_shape_2D = EXTENSIBLE ENUMERATION OF
    (quadrilateral,
     triangle,
     polygon);
END_TYPE;
(*
```

Enumerated item definitions:

quadrilateral: topologically quadrilateral (four sided) requiring a minimum of 4 vertices;

triangle: topologically triangular (three sided) requiring a minimum of 3 vertices;

polygon: topologically polygonal (n-sided) requiring a minimum of 3 vertices;

4.3.5 cell_shape_3D

An identifier of a topologically 3–D unstructured mesh cell shape.

EXPRESS specification:

```
*)
TYPE cell_shape_3D = EXTENSIBLE ENUMERATION OF
    (hexahedron,
     wedge,
     tetrahedron,
     pyramid);
END_TYPE;
(*
```

Enumerated item definitions:

hexahedron: topologically hexahedral (six quadrilateral faces) requiring a minimum of 8 vertices.

wedge: topologically pentahedral (three quadrilateral faces and two triangular faces) requiring a minimum of 6 vertices;

tetrahedron: topologically tetrahedral (four triangular faces) requiring a minimum of 4 vertices;

pyramid: topologically pyramidal (one quadrilateral face and four triangular faces) requiring a minimum of 5 vertices;

4.4 mesh_topology_schema entity definitions

4.4.1 mesh

The basis of all mesh topologies. A **mesh** is a **topological_representation_item** consisting of one or more cells.

EXPRESS specification:

```
*)
ENTITY mesh
  SUBTYPE OF (topological_representation_item);
```



```
  descriptions : LIST OF text;
  index_count  : INTEGER;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_mesh FOR mesh;
  ABSTRACT SUPERTYPE;
  ONEOF(structured_mesh,
        unstructured_mesh);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

descriptions: Annotation;

index_count: The number of indices required to identify uniquely a vertex or cell in the mesh.

NOTE 1 It inherits a **name** attribute of type **label** via its **topological_representation_item** supertype.

4.4.2 topological_region

A **topological_region** is a **topological_representation_item** that is a continuous point set with a single topological dimension.

EXPRESS specification:

```
*)
ENTITY topological_region
  SUBTYPE OF (topological_representation_item);
  descriptions : LIST OF text;
  dimension    : INTEGER;
  boundary     : OPTIONAL BAG OF UNIQUE topological_representation_item;
END_ENTITY;
(*
```

Attribute definitions:

descriptions: Annotation;

dimension: The topological dimension of the region.

boundary: If desired, the elements forming the boundary of the region.

4.4.3 product_of_mesh

A **product_of_mesh** is a relationship that is between:

— two operands that are a 1-dimensional **mesh** and an n-dimensional **mesh**; and

— a product that is an $(n+1)$-dimensional **mesh**,

that indicates the $(n+1)$-dimensional **mesh** is the Cartesian product of the operands.

The ordering of cells and vertices of the product **mesh** is:

— cell $i + n(j-1)$ of the product mesh corresponds to cell i of the first operand and cell j of the second operand, where n is the total number of cells of the first operand;

— vertex $i + m(j-1)$ of the product mesh corresponds to vertex i of the first operand and vertex j of the second operand, where m is the total number of vertices of the first operand.

EXPRESS specification:

```
*)
ENTITY product_of_mesh;
  operands : LIST [2:2] OF mesh;
  product  : mesh;
WHERE
  wr1 : (this_schema+'.STRUCTURED_MESH' IN TYPEOF(operands[1])) AND
        (this_schema+'.STRUCTURED_MESH' IN TYPEOF(operands[2])) AND
        (this_schema+'.STRUCTURED_MESH' IN TYPEOF(product));
  wr2 : operands[1].index_count = 1;
  wr3 : operands[1].index_count + operands[2].index_count
        = product.index_count;
END_ENTITY;
(*
```

Attribute definitions:

operands: the two **mesh**es that define the **product**;

product: the **mesh** that is the Cartesian product of the **operands**.

Formal propositions:

wr1: All meshes shall be **structured_mesh**es;

wr2: The first operand shall have an **index_count** of one;

Table 4 – Number of vertices in a structured_mesh

Index_count	Rectangular	Cylindrical	Pyramidal
1	i	i	i
2	ij	$(1-1)j+1$	$(i-1)j+1$
3	ijk	$(i-1)jk+k$	$(i-1)jk+1$

wr3: the **index_count** of the **product** shall equal the sum of the **index_count**s of the **operands**.

4.4.4 structured_mesh

A mesh topology that is regular.

EXPRESS specification:

```
*)
ENTITY structured_mesh
  SUBTYPE OF (mesh);
  vertex_counts : ARRAY [1:SELF/mesh.index_count] OF INTEGER;
  cell_counts   : ARRAY [1:SELF/mesh.index_count] OF INTEGER;
  rind_planes   : OPTIONAL rind;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_structured_mesh FOR structured_mesh;
  ABSTRACT SUPERTYPE;
  ONEOF(rectangular_grid,
        cylindrical_grid,
        pyramidal_grid);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

vertex_counts: The number of vertices in each dimension of the mesh. The product of the array elements is the number of vertices defining the mesh (i.e., excluding any rind points). The number of vertices in one- two- and three-dimensional regular mesh topologies is given in Table 4, where i, j and k correspond to the array elements `vertex_counts[1]`, `vertex_counts[2]` and `vertex_counts[3]`, respectively.

cell_counts: The number of cells in each dimension of the mesh. The product of the array elements is the number of cells on the interior of the mesh.

index_count: (inherited) The number of indices required to identify uniquely a vertex or cell in the mesh is the same as the topological dimensionality (e.g., 1–D, 3–D) of the mesh.

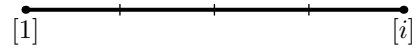


Figure 4 – A 1-D rectangular_grid or cylindrical_grid or pyramidal_grid (with $i = 5$)

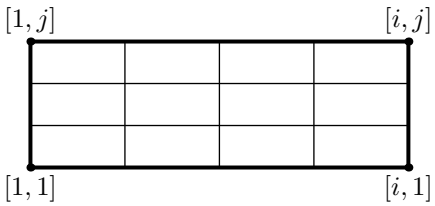


Figure 5 – A 2-D rectangular_grid (with $i = 5$, $j = 4$)

4.4.5 rectangular_grid

A regular mesh topology that is topologically linear in 1–D, quadrilateral in 2–D, hexahedral in 3–D, etc.

NOTE 1 Illustrations of rectangular grid topologies are shown in Figure 4, Figure 5 and Figure 6.

EXPRESS specification:

```
*)
ENTITY rectangular_grid
  SUBTYPE OF (structured_mesh);
END_ENTITY;
(*
```

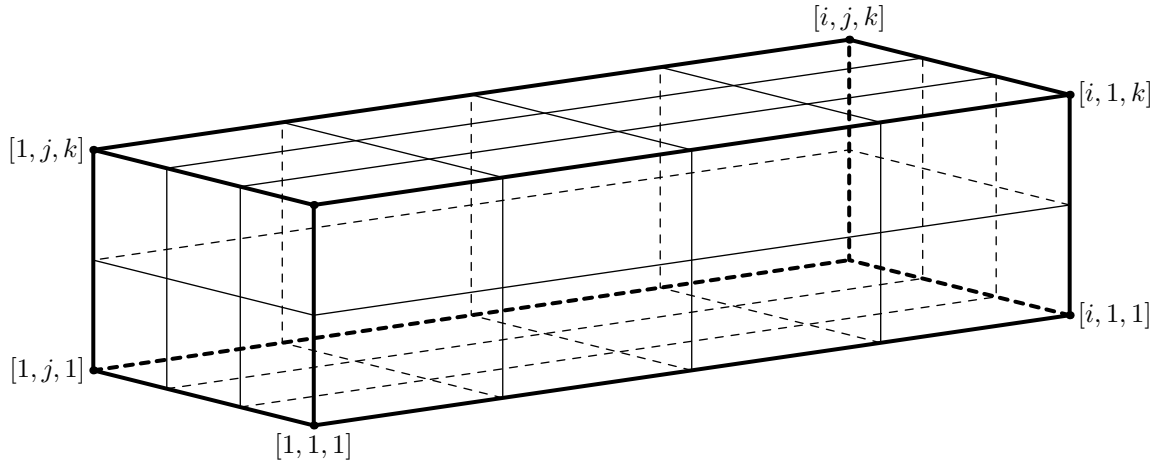


Figure 6 – A 3-D rectangular_grid (with $i = 5$, $j = 4$, $k = 3$)

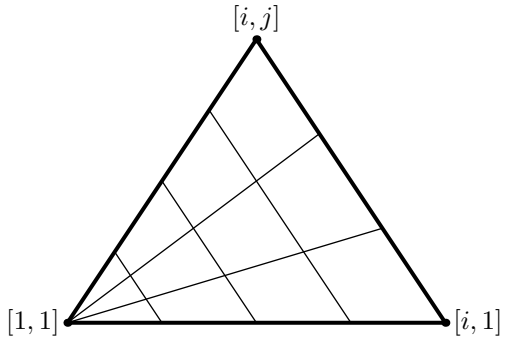


Figure 7 – A 2-D cylindrical_grid or pyramidal_grid (with $i = 5$, $j = 4$)

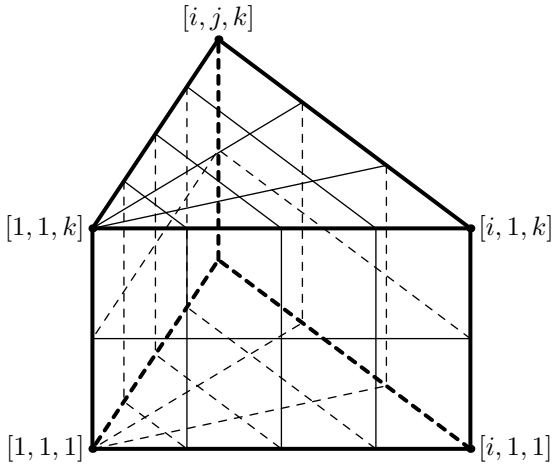


Figure 8 – A 3-D cylindrical_grid (with $i = 5$, $j = 4$, $k = 3$)

4.4.6 cylindrical_grid

A regular mesh topology that is topologically linear in 1–D, a sector of a circle in 2–D (including the centre point), a sector of a cylinder in 3–D (including the axis), etc. The cells are rectangular except for the ring at the centre which are triangular in 2–D and pentahedral in 3–D.

The i direction is in the outward radial direction, the j direction is in the circumferential direction, and the k direction is in the axial direction.

NOTE 1 Illustrations of cylindrical grid topologies are shown in Figure 4, Figure 7 and Figure 8.

EXPRESS specification:

```
*)
ENTITY cylindrical_grid
```

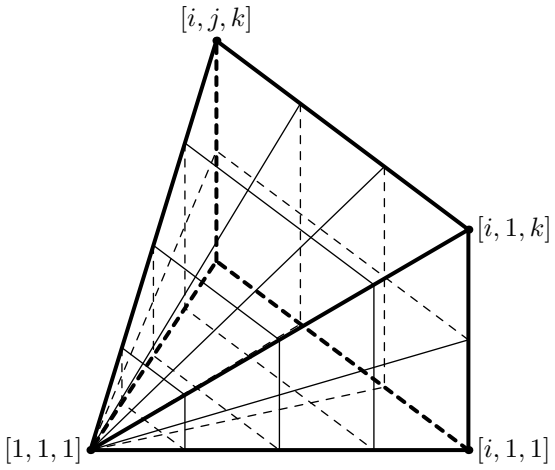



Figure 9 – A 3-D pyramidal_grid (with $i = 5$, $j = 4$, $k = 3$)

```
  SUBTYPE OF (structured_mesh);
END_ENTITY;
(*
```

4.4.7 pyramidal_grid

A regular mesh topology that is topologically linear in 1–D, a sector of a circle in 2–D (including the centre point), a sector of a sphere in 3–D (including the centre point), etc. The cells are rectangular except for the ring at the centre which are triangular in 2–D and pyramidal in 3–D.

The i direction is in the outward radial direction, the j direction is in the longitudinal (equatorial) direction, and the k direction is in latitudinal (polar) direction.

NOTE 1 Illustrations of pyramidal grid topologies are shown in Figure 4, Figure 7 and Figure 9.

EXPRESS specification:

```
*)
ENTITY pyramidal_grid
  SUBTYPE OF (structured_mesh);
END_ENTITY;
(*
```

4.4.8 rind

rind describes the number of rind planes associated with a regular mesh.

EXPRESS specification:

```
*)
ENTITY rind;
  index_count : INTEGER;
  planes      : ARRAY [1:2*index_count] OF INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

index_count: The number of indices required to reference a vertex.

planes: contains the number of rind planes attached to the minimum and maximum faces of a regular mesh. The face corresponding to each index n of **planes** in 3-D is:

$$\begin{aligned} n = 1 &\rightarrow i\text{-min} & \qquad n = 2 &\rightarrow i\text{-max} \\ n = 3 &\rightarrow j\text{-min} & \qquad n = 4 &\rightarrow j\text{-max} \\ n = 5 &\rightarrow k\text{-min} & \qquad n = 6 &\rightarrow k\text{-max} \end{aligned}$$

EXAMPLE 1 For a 3-D grid whose 'core' size is `II`×`JJ`×`KK`, a value of **planes** `= [a,b,c,d,e,f]` indicates that the range of indices for the grid with this rind is:

i: `(1 - a, II + b)`
j: `(1 - c, JJ + d)`
k: `(1 - e, KK + f)`

4.4.9 cell_of_structured_mesh

A **cell_of_structured_mesh** is an identified cell of a **structured_mesh**.

EXPRESS specification:

```
*)
ENTITY cell_of_structured_mesh
  SUBTYPE OF (topological_region);
  the_mesh : structured_mesh;
  cell_identifier : ARRAY [1:index_count] OF INTEGER;
DERIVE
  index_count : INTEGER := the_mesh\mesh.index_count;
END_ENTITY;
(*
```


Attribute definitions:

the_mesh: the **structured_mesh**;

cell_identifier: the indices of the cell;

index_count: the number of indices required to uniquely identify a vertex or cell in the mesh;

4.4.10 composition_of_structured_mesh

A **composition_of_structured_mesh** is a relationship between two **structured_mesh**es that indicates one is part of the other.

EXPRESS specification:

```
*)
ENTITY composition_of_structured_mesh;
  part         : structured_mesh;
  whole        : structured_mesh;
  lower_vertex : ARRAY [1:whole_indices] OF INTEGER;
  lower_face   : ARRAY [1:whole_indices] OF OPTIONAL BOOLEAN;
  used_indices : ARRAY [1:part_indices] OF INTEGER;
  used_senses  : ARRAY [1:part_indices] OF BOOLEAN;
DERIVE
  whole_indices : INTEGER := whole\mesh.index_count;
  part_indices  : INTEGER := part\mesh.index_count;
END_ENTITY;
(*
```

Attribute definitions:

part: the **structured_mesh** that is part of the **whole**;

whole: the **structured_mesh** that contains the **part**;

lower_vertex: The position of the **vertex** in the **whole** that is the origin of the **part**. This is specified with respect to each index of the **whole**.

used_indices: the indices of the **whole** that are also indices of the **part** in the order that they are used in the **part**;

used_senses: the sense for each index of **part** as:

— TRUE if the **part** uses the index of the **whole** in the same direction;

— FALSE if the **part** uses the index of the **whole** in the reverse direction;

whole_indices: the number of indices required to uniquely identify a vertex or cell in the **whole**;

part_indices: the number of indices required to uniquely identify a vertex or cell in the **part**;

4.4.11 unstructured_mesh

A mesh topology that is not regular. It conceptually consists of the vertices of the mesh and the cells forming the volume of the mesh. The cells shall all be connected by each cell having at least one vertex in common with another cell.

EXPRESS specification:

```
*)
ENTITY unstructured_mesh
  SUBTYPE OF (mesh);
  vertex_count : INTEGER;
  cell_count   : INTEGER;
  cells        : ARRAY [1:cell_count] OF topological_region;
  vertices     : OPTIONAL ARRAY [1:vertex_count] OF vertex;
WHERE
  oned : SELF/mesh.index_count = 1;
END_ENTITY;
(*
```

Attribute definitions:

vertex_count: is the number of vertices in the mesh. The vertex indices range from 1 to **vertex_count**.

cell_count: is the number of cells in the mesh;

cells: is the **topological_regions** forming the mesh.

vertices: if required, are the **vertex**es referenced by the **cells**.

index_count: (inherited) the number of indices required to uniquely identify a vertex or cell in the mesh;

Formal propositions:

oned: The value of **index_count** shall be 1.

4.4.12 vertex_defined_cell

A **vertex_defined_cell** is a **topological_region** that is bounded by vertices; the number of vertices depends on the topological shape of the cell. The cell may have interior nodes; the maximum number of interior nodes depends on both the shape and the order of the cell.

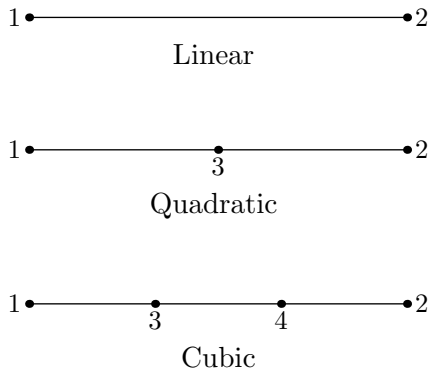


Figure 10 – Linear, quadratic and cubic bar cells

EXPRESS specification:

```
*)
ENTITY vertex_defined_cell
 SUBTYPE OF (topological_region);
  shape       : cell_shape;
  order       : element_order;
  vn_count    : INTEGER;
  vertices    : ARRAY [1:vn_count] OF INTEGER;
  nodes       : OPTIONAL ARRAY [1:opt_node_count] OF OPTIONAL INTEGER;
DERIVE
  bound_count     : positive := cell_counts(SELF)[1];
  edge_node_count : positive := cell_counts(SELF)[2];
  opt_node_count  : positive := cell_counts(SELF)[3];
WHERE
  wr1 : ((NOT EXISTS(nodes)) AND (opt_node_count <= 0));
END_ENTITY;
(*
```

Attribute definitions:

description: annotation;

shape: the topological shape of the cell;

order: the order of the cell geometric interpolation;

vn_count: the number of bounding vertices plus the number of edge nodes;

vertices: is the indices of the cell vertices and edge nodes. The position of a vertex or an edge node in the array depends on the shape of the cell as established graphically in Figure 10 to Figure 16, where a vertex or edge node is indicated by a dot. The vertex labelled '1' is the first index in the array, that labelled '2' is the second index in the array, and so on.

nodes: is the indices of the cell's non-edge interior nodes. If a cell has no interior nodes then

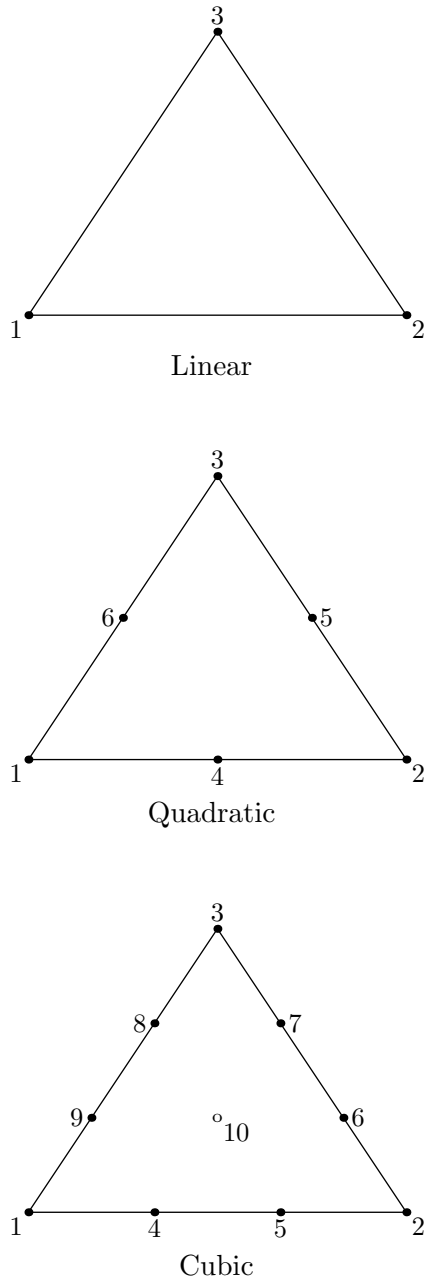


Figure 11 – Linear, quadratic and cubic triangle cells

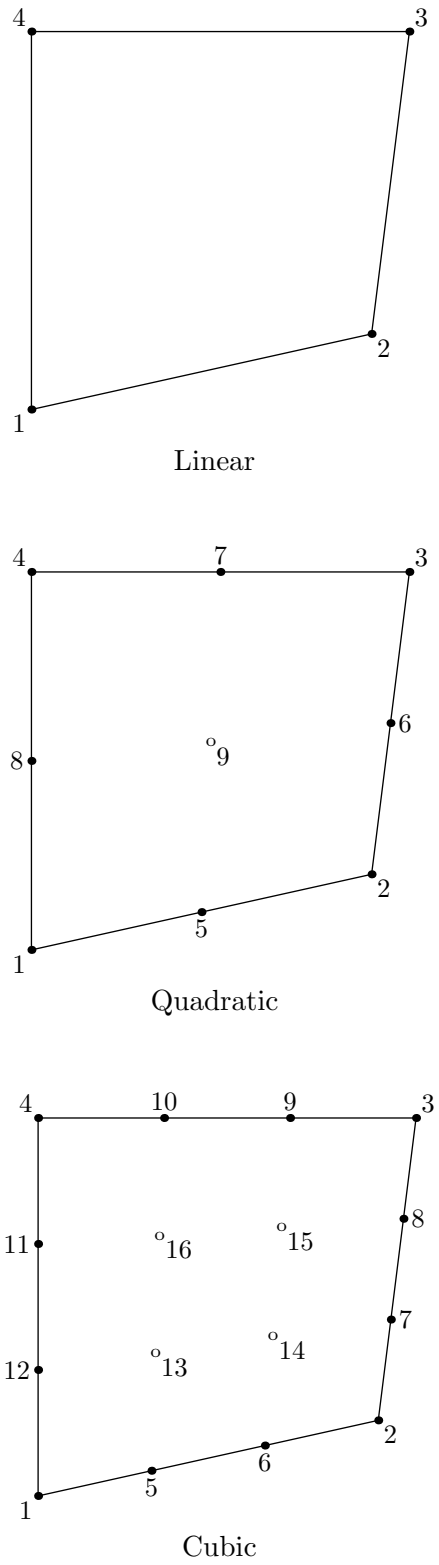
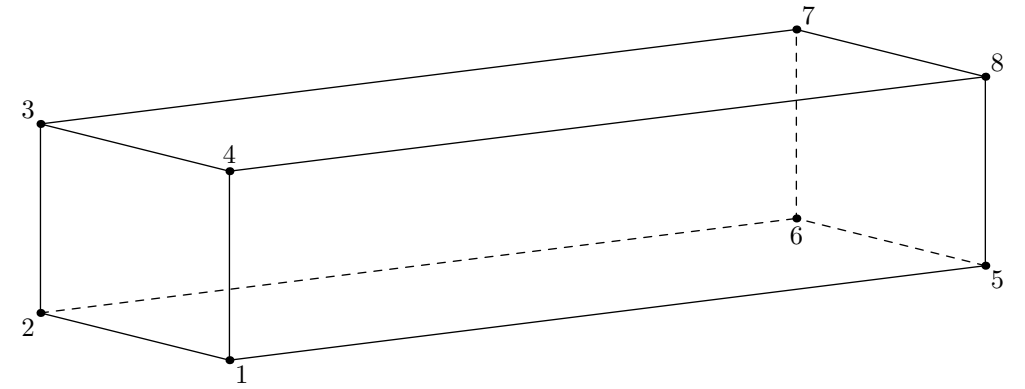
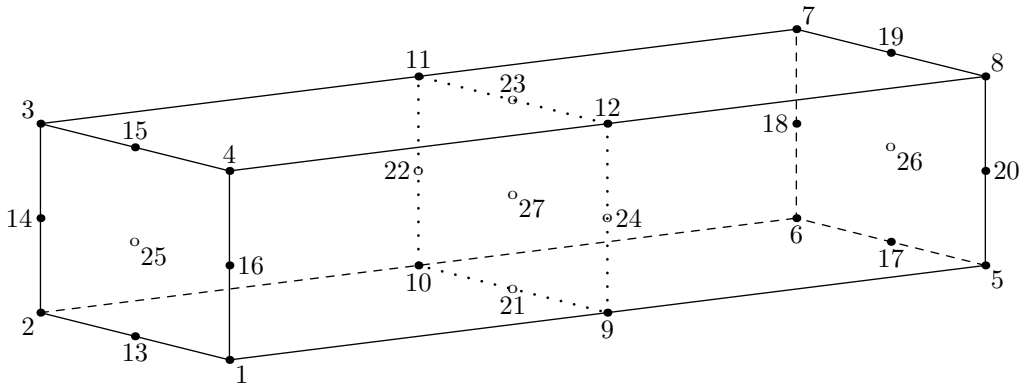


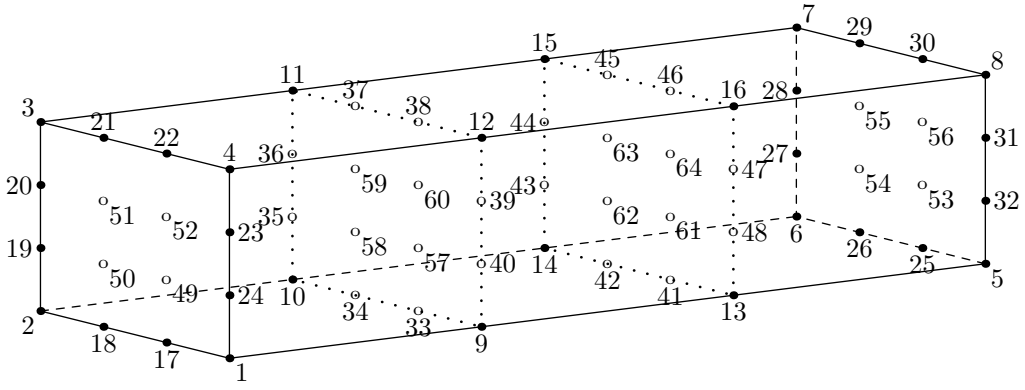
Figure 12 – Linear, quadratic and cubic quadrilateral cells



Linear

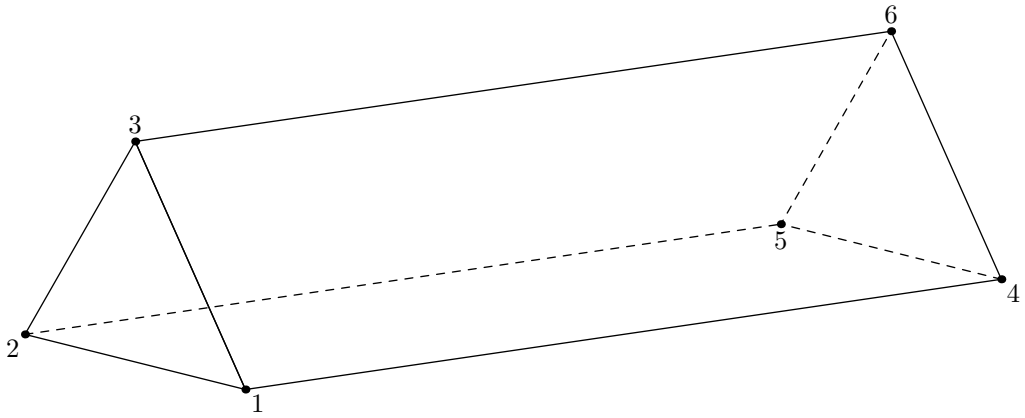


Quadratic

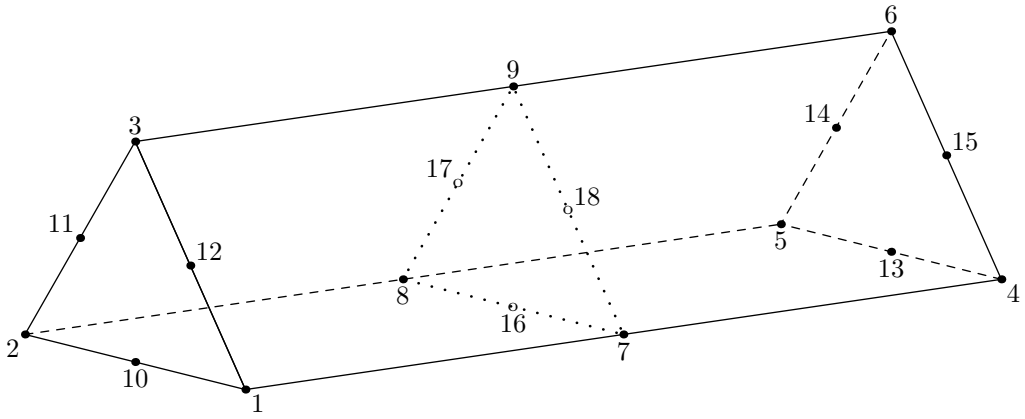


Cubic

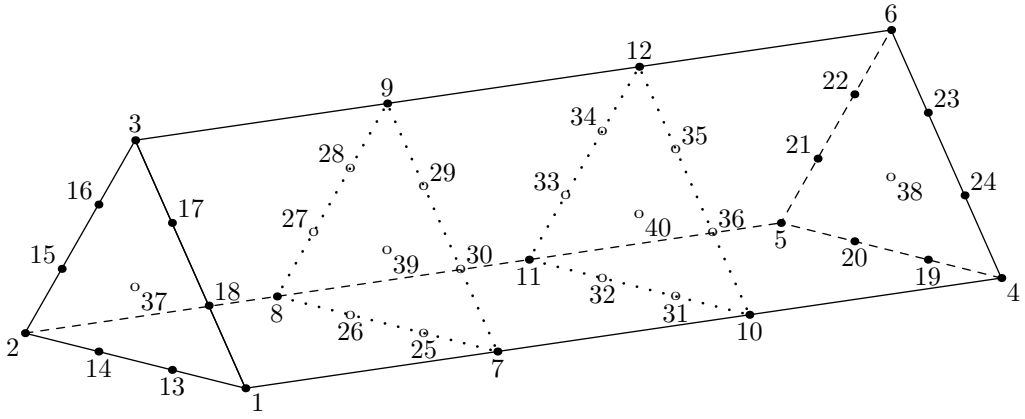
Figure 13 – Linear, quadratic and cubic hexahedron cells



Linear



Quadratic



Cubic

Figure 14 – Linear, quadratic and cubic wedge cells

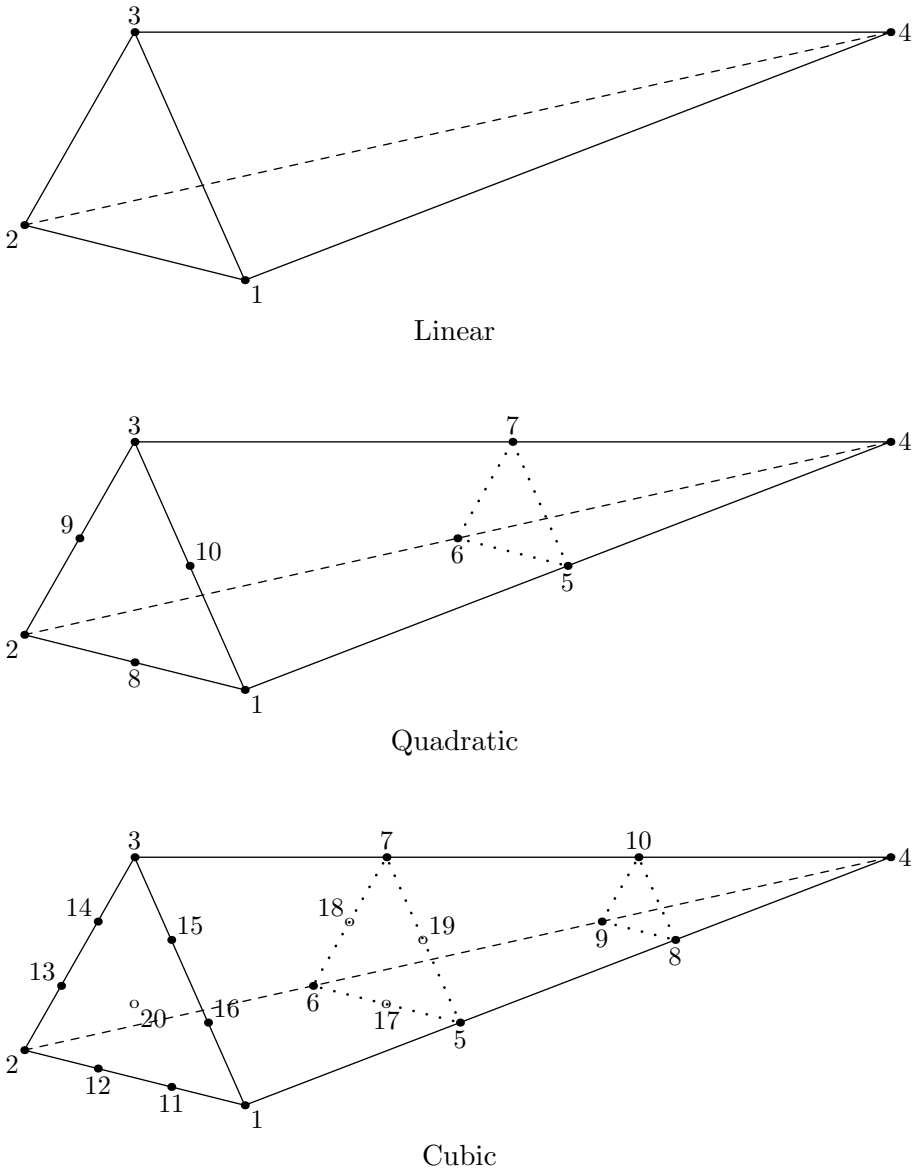
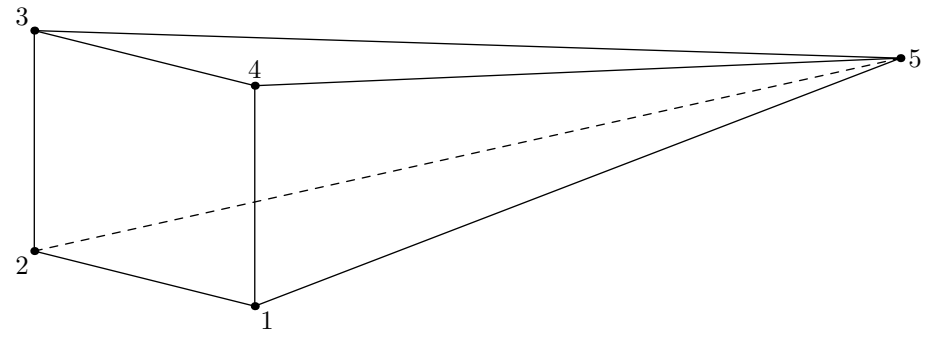
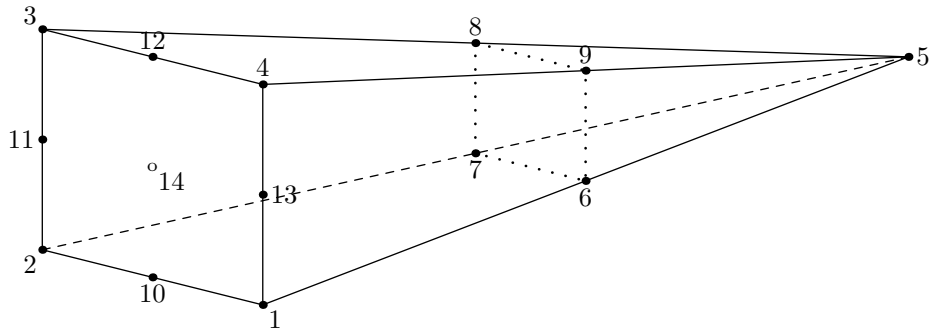


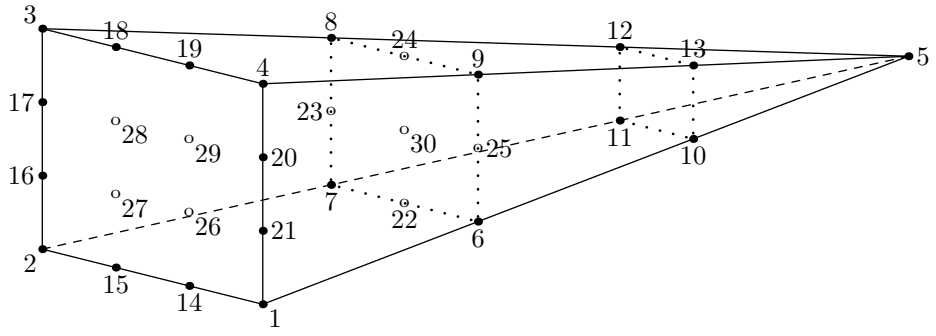
Figure 15 – Linear, quadratic and cubic tetrahedron cells



Linear



Quadratic



Cubic

Figure 16 – Linear, quadratic and cubic pyramid cells

the attribute shall have no value. The ordering of the indices of the interior nodes depends on the shape and order of the cell as established graphically in Figure 10 to Figure 16, where a non-edge interior node is indicated by a circle. The node labelled $v + 1$, where v is the value of **vn_count**, is the first index in the array, that labelled $v + 2$ is the second index in the array, and so on. If a particular interior node is not supported by the cell, that position in the array shall be given the value `?`.

bound_count: is the number of cell bounding vertices; it is determined by the value of **shape**;

edge_node_count: is the number of interior cell nodes located on the cell edges; it is determined by the combination of the values of **shape** and **order**;

opt_node_count: is the potential number of interior cell nodes which are not located on the cell edges; it is determined by the combination of the values of **shape** and **order**;

NOTE 1 The maximum total number of potential geometric locations is given by the sum of **bound_count**, **edge_node_count** and **opt_node_count**.

Formal propositions:

wr1: If the **opt_node_count** is zero or less then **nodes** shall have no value.

4.4.13 mesh_data

mesh_data associates data values with a **mesh**.

EXPRESS specification:

```
*)
ENTITY mesh_data
  ABSTRACT;
  descriptions : LIST OF text;
  id           : label;
  the_mesh     : mesh;
  data         : GENERIC;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_mesh_data FOR mesh_data;
  ABSTRACT SUPERTYPE;
  ONEOF(mesh_cell_data,
        mesh_vertex_data);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

descriptions: annotation;

id: an identifier;

the_mesh: The **mesh**;

data: The data.

4.4.14 mesh_cell_data

mesh_cell_data associates data values with the cells of a **mesh**.

EXPRESS specification:

```
*)
ENTITY mesh_cell_data
  SUBTYPE OF (mesh_data);
  SELF/mesh_data.data : data_array;
WHERE
  wr1 : SELF/mesh_data.the_mesh.index_count = data.dimension;
END_ENTITY;
(*
```

Attribute definitions:

the_mesh: (inherited) the **mesh**;

data: (inherited) the cell data.

Formal propositions:

wr1: The **index_count** of the **mesh** and the **dimension** of the **data** shall have the same value.

4.4.15 mesh_vertex_data

mesh_vertex_data associates data values with the vertices of a **mesh**.

EXPRESS specification:

```
*)
ENTITY mesh_vertex_data
  SUBTYPE OF (mesh_data);
  SELF/mesh_data.data : data_array;
WHERE
  wr1 : SELF/mesh_data.the_mesh.index_count = data.dimension;
END_ENTITY;
```



```
(*
```

Attribute definitions:

the_mesh: (inherited) the **mesh**;

data: (inherited) the vertex data.

Formal propositions:

wr1: The **index_count** of the **mesh** and the **dimension** of the **data** shall have the same value.

4.5 mesh_topology_schema function definitions

4.5.1 this_schema

The function **this_schema** returns a STRING containing the name of the schema.

EXPRESS specification:

```
*)
FUNCTION this_schema : STRING;
  RETURN('MESH_TOPOLOGY_SCHEMA');
END_FUNCTION;
(*
```

Argument definitions:

RETURNS: The uppercase name of the schema.

4.5.2 cell_counts

cell_counts takes a **vertex_defined_cell** as its argument and returns the numbers of vertices and nodes required to define the cell.

EXPRESS specification:

```
*)
FUNCTION cell_counts(arg : vertex_defined_cell) : ARRAY[1:3] OF INTEGER;
LOCAL
  om1    : INTEGER := 0;      -- (order - 1)
  om1sq  : INTEGER := om1**2; -- (order - 1) squared
```



```
  vts    : INTEGER;           -- number of bounding vertices
  eds    : INTEGER;           -- number of edges
  qf     : INTEGER := 0;      -- number of quadrilateral faces
  tf     : INTEGER := 0;      -- number of triangular faces
  result : ARRAY [1:3] OF INTEGER := [0,0,0];
END_LOCAL;
  CASE arg.order OF
    linear    : om1 := 0;
    quadratic : om1 := 1;
    cubic     : om1 := 2;
    OTHERWISE : RETURN(result);
  END_CASE;
  om1sq := om1**2;
  CASE arg.shape OF
    single :
        BEGIN
          vts := 1; eds := 0; qf := 0; tf := 0;
          result[1] := vts;
          result[2] := om1*eds;                  -- 0, 0, 0
          result[3] := 0;                        -- 0, 0, 0
        END;
    bar :
        BEGIN
          vts := 2; eds := 1; qf := 0; tf := 0;
          result[1] := vts;
          result[2] := om1*eds;                  -- 0, 1, 2
          result[3] := 0;                        -- 0, 0, 0
        END;
    quadrilateral :
        BEGIN
          vts := 4; eds := 4; qf := 1; tf := 0;
          result[1] := vts;
          result[2] := om1*eds;                  -- 0, 4, 8
          result[3] := om1sq*qf;                 -- 0, 1, 4
        END;
    triangle :
        BEGIN
          vts := 3; eds := 3; qf := 0; tf := 1;
          result[1] := vts;
          result[2] := om1*eds;                  -- 0, 3, 6
          result[3] := (om1-1)*tf;               --    0, 1
          CASE arg.order OF
            linear : result[3] := 0;             -- 0
          END_CASE;
        END;
    polyhedron :
        BEGIN
          vts := arg.vn_count; eds := arg.vn_count;
          result[1] := vts;
          result[2] := 0;
          result[3] := 0;
        END;
```



```
    hexahedron :
        BEGIN
          vts := 8; eds := 12; qf := 6; tf := 0;
          result[1] := vts;
          result[2] := om1*eds;                    -- 0, 12, 24
          result[3] := om1sq*(qf+om1);             -- 0, 7, 32
        END;
    wedge :
        BEGIN
          vts := 6; eds := 9; qf := 3; tf := 2;
          result[1] := vts;
          result[2] := om1*eds;                    -- 0, 9,  18
          result[3] := om1sq*qf + om1*tf;          -- 0, 3, 16
        END;
    tetrahedron :
        BEGIN
          vts := 4; eds := 6; qf := 0; tf := 4;
          result[1] := vts;
          result[2] := om1*eds;                    -- 0, 6, 12
          result[3] := (om1-1)*tf;                 --    0, 4
          CASE arg.order OF
            linear : result[3] := 0;               -- 0
          END_CASE;
        END;
    pyramid :
        BEGIN
          vts := 5; eds := 8; qf := 1; tf := 4;
          result[1] := vts;
          result[2] := om1*eds;                    -- 0, 8, 16
          result[3] := om1sq*qf + (om1-1)*tf;      --    1, 9
          CASE arg.order OF
            linear : result[3] := 0;               -- 0
          END_CASE;
        END;
  END_CASE;
RETURN(result);
END_FUNCTION;
(*
```

Argument definitions:

arg: A **cell**;

RETURNS: A 3 element array of INTEGER, where the first element is the number of vertices defining the bounds of the cell, the second is the number of interior nodes located on an edge, and the third is the maximum number of (potential) interior nodes not located on an edge.

EXPRESS specification:

```
*)
END_SCHEMA;  -- end of mesh_topology_schema
(*
```

5 data_array_schema

The following EXPRESS declaration begins the **data_array_schema** and identifies the necessary external references.

EXPRESS specification:

```
*)
{iso standard 10303 part (11) version (4)}
SCHEMA data_array_schema;
  REFERENCE FROM measure_schema
    (dimensional_exponents,
     named_unit);
  REFERENCE FROM support_resource_schema
    (label,
     text);
  REFERENCE FROM mathematical_functions_schema
    (explicit_table_function);
(*
```

NOTE The schemas referenced above can be found in the following parts of ISO 10303:

measure_schema	ISO 10303-41
support_resource_schema	ISO 10303-41
mathematical_functions_schema	ISO 10303-50

5.1 Introduction

This schema defines and describes the structures and types that may be used to contain data values.

5.2 Fundamental concepts and assumptions

The structure type **data_array** is a general purpose structure for holding arrays of data which can be represented by the EXPRESS simple data types. Every element in a **data_array** shall be of the same data type. It may be used to describe, for example, mesh coordinates or any other information, whether or not it is related the mesh topologies.

Five classes of numeric data are addressed with the **data_array** structure type:

a) dimensional data (e.g., velocity in units of m/s);

b) nondimensional data normalized by dimensional reference quantities;

c) nondimensional data with associated nondimensional reference quantities;

d) nondimensional parameters (e.g., Reynolds number, pressure coefficient);

e) pure constants (e.g., π, e).

Each of the five classes of numeric data requires different information to describe dimensional units or normalization associated with the data.

Identifiers or names can be attached to **data_array** entities to identify and describe the quantity being stored. To facilitate communication between different applications, provision is made for a set, or sets, of standardized data-name identifiers. For any identifier in these sets, the associated data should be unambiguously understood. For coordinate data a list of standardized identifiers is provided. For example, the standardized identifier **coordinate_y** shall be used for data arrays containing values representing Y-coordinates in a Cartesian coordinate system.

All standardized identifiers denote scalar quantities; this is consistent with the intended use of the **data_array** structure type to describe an array of scalars. For quantities that are vectors, such as velocity, their components are listed.

Included with the lists of standard data-name identifiers, the fundamental units of dimensions associated with that quantity are provided. The following notation is used for the fundamental units: **M** is mass, **L** is length, **T** is time, Θ is temperature and α is angle. These fundamental units are directly associated with the elements of the **DimensionalExponents_t** structure. For example, a quantity that has dimensions **ML/T** corresponds to **MassExponent** = +1, **LengthExponent** = +1, and **TimeExponent** = -1.

5.3 data_array_schema type definitions

5.3.1 data_class

data_class is an enumeration type that identifies the class of a given piece of data.

EXPRESS specification:

```
*)
TYPE data_class = EXTENSIBLE ENUMERATION OF
     (unspecified,
      user_defined,
      dimensional,
      normalized_by_dimensional,
```



```
      normalised_by_unknown_dimensional,
      nondimensional_parameter,
      dimensionless_constant);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: No identification;

user_defined: meaning is assigned external to this standard;

dimensional: Identifies dimensional data;

normalized_by_dimensional: Identifies nondimensional data that is normalised by dimensional reference quantities;

normalised_by_unknown_dimensional: Identifies nondimensional data typically found in completely nondimensional object base where all field and reference data is nondimensional;

nondimensional_parameter: Identifies nondimensional parameters like Mach number or lift coefficient;

dimensionless_constant: Identifies constants like π.

5.3.2 data_name

data_name is an identifier for the contents of a **data_array**. It is a superset of **standard_data_name** and **adhoc_data_name**.

EXPRESS specification:

```
*)
TYPE data_name = SELECT
     (standard_data_name,
      adhoc_data_name);
END_TYPE;
(*
```

5.3.3 adhoc_data_name

adhoc_data_name is a STRING providing a non-standard identifier for the contents of a **data_array**.

EXPRESS specification:

```
*)
```



```
TYPE adhoc_data_name = STRING;
END_TYPE;
(*
```

5.3.4 standard_data_name

standard_data_name is a listing of standardized identifiers for the contents of a **data_array**.

EXPRESS specification:

```
*)
TYPE standard_data_name = EXTENSIBLE SELECT
     (coordinate_data_name,
      other_data_name);
END_TYPE;
(*
```

5.3.5 coordinate_data_name

coordinate_data_name is an enumeration of standardized coordinate systems data.

Coordinate systems for identifying physical location are as follows:

System	3–D	2–D
Cartesian	(x, y, z)	(x, y) or (x, z) or (y, z)
Cylindrical	(r, θ, z)	(r, θ)
Spherical	(r, θ, ϕ)	
Auxiliary	(ξ, η, ζ)	(ξ, η) or (ξ, ζ) or (η, ζ)

Associated with these coordinate systems are the following unit vector conventions:

x-direction $\hat{e}_x$	r-direction $\hat{e}_r$	ξ-direction $\hat{e}_\xi$
y-direction $\hat{e}_y$	θ-direction $\hat{e}_\theta$	η-direction $\hat{e}_\eta$
z-direction $\hat{e}_z$	ϕ-direction $\hat{e}_\phi$	ζ-direction $\hat{e}_\zeta$

Note that $\hat{e}_r$, $\hat{e}_\theta$ and $\hat{e}_\phi$ are functions of position.

It is expected that one of the 'standard' coordinate systems (cartesian, cylindrical or spherical) will be used within a mesh (or perhaps the entire object base) to define grid coordinates and other related data. The auxiliary coordinates may be used for special quantities, such as forces and moments, which may not be defined in the same coordinate system as the rest of the data. When auxiliary coordinates are used, a transformation shall also be provided to uniquely define them.

Table 5 – Coordinate data name identifiers

Data name identifier	Description	Units
coordinate_x	x	**L**
coordinate_y	y	**L**
coordinate_z	z	**L**
coordinate_r	r	**L**
coordinate_theta	θ	α
coordinate_phi	ϕ	α
coordinate_normal	coordinate in direction of $\hat{e}_n$	**L**
coordinate_tangential	tangential coordinate (2–D only)	**L**
coordinate_xi	ξ	**L**
coordinate_eta	η	**L**
coordinate_zeta	ζ	**L**
coordinate_transform	transformation matrix (**T**)	—

EXAMPLE 1 The transform from cartesian to orthogonal auxiliary coordinates is,

$$\begin{pmatrix} \hat{e}_\xi \\ \hat{e}_\eta \\ \hat{e}_\zeta \end{pmatrix} = \mathbf{T} \begin{pmatrix} \hat{e}_x \\ \hat{e}_y \\ \hat{e}_z \end{pmatrix},$$

where **T** is an orthonormal matrix (2×2 in 2–D and 3×3 in 3–D).

In addition, normal and tangential coordinates are often used to define boundary conditions and data related to surfaces. The normal coordinate is identified as n with the unit vector $\hat{e}_n$.

EXPRESS specification:

```
*)
TYPE coordinate_data_name = EXTENSIBLE ENUMERATION OF
     (coordinate_x,
      coordinate_y,
      coordinate_z,
      coordinate_r,
      coordinate_theta,
      coordinate_phi,
      coordinate_normal,
      coordinate_tangential,
      coordinate_xi,
      coordinate_eta,
      coordinate_zeta,
      coordinate_transform);
END_TYPE;
(*
```

The meanings of the enumerated coordinate data identifiers are given in Table 5.

5.3.6 other_data_name

other_data_name is an enumeration of standardized data identifiers.

EXPRESS specification:

```
*)
TYPE other_data_name = EXTENSIBLE ENUMERATION OF
     (unspecified,
      nondimensional);
END_TYPE;
(*
```

Enumerated item definitions:

unpsecified: Not specified;

nondimensional: Non-dimensional data.

5.4 data_array_schema entity definitions

5.4.1 data_conversion

data_conversion contains conversion factors for recovering raw dimensional data from given nondimensional data, or for linear conversion of values in one set of units to the equivalent values in another set of compatible units.

Given a nondimensional piece of data, `Data(nondimensional)`, the conversion to 'raw' dimensional form is:

```
Data(raw) = Data(nondimensional)*scale + offset
```

EXAMPLE 1 The **data_conversion** from Fahrenheit temperature units to Celsius temperature units is:

```
c2f := data_conversion(5/9, -((5*32)/9));
```

EXPRESS specification:

```
*)
ENTITY data_conversion;
  scale  : REAL;
  offset : REAL;
END_ENTITY;
```



```
(*
```

Attribute definitions:

scale: The scaling factor.

offset: The offset.

5.4.2 dimensional_units

dimensional_units describes the system of units used to measure dimensional data.

If all the attributes have the value **Null**, that is equivalent to saying that the data described by that instance is nondimensional.

EXPRESS specification:

```
*)
ENTITY dimensional_units;
  length              : OPTIONAL named_unit;
  mass                : OPTIONAL named_unit;
  time                : OPTIONAL named_unit;
  current             : OPTIONAL named_unit;
  temperature         : OPTIONAL named_unit;
  amount_of_substance : OPTIONAL named_unit;
  luminous_intensity  : OPTIONAL named_unit;
  plane_angle         : OPTIONAL named_unit;
  solid_angle         : OPTIONAL named_unit;
END_ENTITY;
(*
```

Attribute definitions:

length: The unit of length;

mass: The unit of mass;

time: The unit of time;

current: The unit of electric current;

temperature: The unit of thermodynamic temperature;

amount_of_substance: The unit of the amount of substance;

luminous_intensity: The unit of luminous intensity;

plane_angle: The unit of plane angle;

solid_angle: The unit of solid angle.

5.4.3 index_list

index_list specifies a list of indices.

EXPRESS specification:

```
*)
ENTITY index_list;
  nindices   : INTEGER;
  indices    : LIST [1:?] OF ARRAY [1:nindices] OF INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

nindices: The number of indices to map to a unique array location.

indices: The indices.

5.4.4 index_range

index_range specifies the beginning and ending indices of a subrange.

EXPRESS specification:

```
*)
ENTITY index_range;
 nindices    : INTEGER;
 start       : ARRAY [1:dimension] OF INTEGER;
 finish      : ARRAY [1:dimension] OF INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

nindices: The number of indices required to identify an element of the range.

start: The indices of the minimal corner of the subrange;

finish: The indices of the maximal corner of the subrange.

5.4.5 data_array

data_array describes a multi-dimensional data array of a given type, dimensionality and size in each dimension. The data may be dimensional, nondimensional or pure constants. Qualifiers are provided to describe dimensional units or normalization information associated with the data.

This structure is formulated to describe an array of scalars. Therefore, for vector quantities (e.g., a position vector or a velocity vector), seperate instances are required for each component of the vector.

EXAMPLE 1 The cartesian coordinates of a 3-D structured mesh are described by three separate data arrays: one for x, one for y, and one for z.

The optional attributes of **data_array** provide information for manipulating the data, including changing units or normalization. Within a given instance of **data_array**, the class of data and all information required for manipulations may be completely and precisely specified by the values of **class**, **units**, **exponents** and **conversion**. **class** identifies the class of data and governs the manipulations that can be performed.

— **dimensional:** When **class** = **dimensional**, the data is dimensional. The optional qualifiers **units** and **exponents** describe dimensional units associated with the data. These qualifiers are provided to specify the system of dimensional units and the dimensional exponents, respectively.

EXAMPLE 2 If the data is the x-component of velocity, then **units** will state that the pertinent dimensional units are, say, **metre** and **second**; **exponents** will specify that the pertinent dimensional exponents are **length** = 1 and **time** = -1. Combining the information gives the units *m/s*.

If **exponents** is absent, then the appropriate dimensional exponents can determined by convention provided the value of **identifier** is one of the **standard_data_name** identifiers, otherwise the exponents are unspecified.

— **normalized_by_dimensional:** When **class** = **normalized_by_dimensional**, the data is nondimensional and is normalized by dimensional reference quantities. All optional entities in **data_array** are used. **conversion** contains factors to convert the nondimensional data to 'raw' dimensional data; these factors are **scale** and **offset**. The conversion process is as follows:

```
Data(raw) = Data(nondimensional)*scale + offset
```

where `Data(nondimensional)` is the original nondimensional data, and `Data(raw)` is the converted raw data. This converted raw data is dimensional, and the optional qualifiers **units** and **exponents** describe the appropriate dimensional units and exponents. Note that **units** and **exponents** also describe the units for **scale** and **offset**.

If **conversion** is absent, the equivalent defaults are **scale** = 1 and **offset** = 0. If either

units or **exponents** is absent, follow the rules described for Dimensional data above.

— **normalized_by_unknown_dimensional:** When **class** = **normalized_by_unknown_dimensional**, the data is nondimensional and is normalized by some unspecified dimensional quantities. This type of data is typical of a completely nondimensional test case, where all field data and all reference quantities are nondimensional.

Only the **exponents** qualifier is used in this case, although it is expected that this qualifier will be seldom utilized in practice. For entities of **data_array** that are not among the list of standardized data-name identifiers, the qualifier could provide useful information by defining the exponents of the dimensional form of the nondimensional data.

Rather than providing qualifiers to describe the normalization of the data, all data of type **normalized_by_unknown_dimensional** in a given object base shall be nondimensionalized consistently. This is done by picking one set of mass, length, time and temperature scales and normalizing all appropriate data by these scales. This process is described in detail in the following.

NOTE 1 The practice of nondimensionalization within flow solvers and other application codes is quite popular. The problem with this practice is that to manipulate the data from a given code, one must often know the particulars of the nondimensionalization used. This largely results from what can be termed inconsistent normalization — more than the minimum required scales are used to normalize data within the code.

EXAMPLE 3 In one CFD flow solver, the following nondimensionalization is used:

$$\begin{array}{lll} \tilde{x} = x/L, & \tilde{u} = u/c_\infty, & \tilde{\rho} = \rho/\rho_\infty, \\ \tilde{y} = y/L, & \tilde{v} = v/c_\infty, & \tilde{p} = p/(\rho_\infty c_\infty^2), \\ \tilde{z} = z/L, & \tilde{w} = w/c_\infty, & \tilde{\mu} = \mu/\mu_\infty, \end{array}$$

where (x, y, z) are the cartesian coordinates, (u, v, w) are the cartesian components of velocity, ρ is static density, p is static pressure, c is the static speed of sound, and μ is the molecular viscosity. In this example, tilde quantities $\tilde{()}$ are nondimensional and all others are dimensional. Four dimensional scales are used for normalization: L (a unit length), ρ_∞, c_∞ and μ_∞. However, only three fundamental dimensional units are represented: mass, length and time. The extra normalizing scale leads to inconsistent normalization. The primary consequence of this is additional nondimensional parameters, such as Reynolds number, appearing in the nondimensionalized governing equations where none are found in the original dimensional equations. Many definitions, including skin friction coefficient, also have extra terms appearing in the nondimensionalized form. This adds unnecessary complication to any data or equation manipulation associated with the flow solver.

Consistent normalization avoids many of these problems. Here the number of scales used for normalization is the same as the number fundamental dimensional units represented by the data. Using consistent normalization, the resulting nondimensionalized form of equations and definitions is identical to their original dimensional formulations. One piece of evidence to support this assertion is that it is not possible to form any nondimensional parameters from the set of dimensional scales used for normalization.

An important fallout of consistent normalization is that the actual scales used for normalization become immaterial for all data manipulation processes. To illustrate this consider the following nondimensionalization procedure: Let M (mass), L (length) and T (time) be arbitrary dimensional

scales by which all data is normalized (neglect temperature data for the present). The nondimensional data follows:

$$\begin{array}{lll} x' = x/L, & u' = u/(L/T), & \rho' = \rho/(M/L^3), \\ y' = y/L, & v' = v/(L/T), & p' = p/(M/(LT^2)), \\ z' = z/L, & w' = w/(L/T), & \mu' = \mu/(M/(LT)), \end{array}$$

where primed quantities are nondimensional and all others are dimensional.

Consider an existing object base where all field data and all reference data is nondimensional and normalized as shown. Assume the object base has a single reference state given by,

$$\begin{array}{lll} x'_{\mathrm{ref}} = x_{\mathrm{ref}}/L, & u'_{\mathrm{ref}} = u_{\mathrm{ref}}/(L/T), & \rho'_{\mathrm{ref}} = \rho_{\mathrm{ref}}/(M/L^3), \\ y'_{\mathrm{ref}} = y_{\mathrm{ref}}/L, & v'_{\mathrm{ref}} = v_{\mathrm{ref}}/(L/T), & p'_{\mathrm{ref}} = p_{\mathrm{ref}}/(M/(LT^2)) \\ z'_{\mathrm{ref}} = z_{\mathrm{ref}}/L, & w'_{\mathrm{ref}} = w_{\mathrm{ref}}/(L/T), & \mu'_{\mathrm{ref}} = \mu_{\mathrm{ref}}/(M/(LT)). \end{array}$$

If a user wanted to change the nondimensionalization of grid-point pressures, the procedure is straightforward. Let the desired new normalization be given by $p''_{ijk} = p_{ijk}/(\rho_{\mathrm{ref}} c^2_{\mathrm{ref}})$, where all terms on the right-hand-side are *dimensional*, and as such they are unknown to the object base user. However, the desired manipulation is possible using only nondimensional data provided in the object base,

$$\begin{aligned} p''_{ijk} &\equiv p_{ijk}/(\rho_{\mathrm{ref}} c^2_{\mathrm{ref}}) \\ &= \frac{p_{ijk}}{M/(LT^2)} \frac{M/L^3}{\rho_{\mathrm{ref}}} \left[\frac{L/T}{c_{\mathrm{ref}}}\right]^2 \\ &= p'_{ijk}/(\rho'_{\mathrm{ref}} (c'_{\mathrm{ref}})^2) \end{aligned}$$

Thus, the desired renormalization is possible using the object base's nondimensional data as if it were actually dimensional. There is, in fact, a high degree of equivalence between dimensional data and consistently normalized nondimensional data. The procedure shown in this example should extend to any desired renormalization, provided the needed reference-state quantities are included in the object base.

This example points out two requirements for data in the class **normalized_by_unknown_dimensional**,

a) All nondimensional data within a given object base that has **class** = **normalized_by_unknown_dimensional** shall be consistently normalized.

b) Any nondimensional reference state appearing in a object base should be sufficiently populated with reference quantities to allow for renormalization procedures.

The second of these stipulations is somewhat ambiguous, but good practice would suggest that a flow solver, for example, should output to the object base all static and stagnation reference quantities defined in the code.

These two stipulations lead to the following:

- The dimensional scales used to nondimensionalize all data are immaterial, and there is no need to identify these quantities in an object base.

- The dimensional scales need not be reference-state quantities provided in the object base. For example, a given object base could contain freestream reference state conditions, but all the data is normalized by sonic conditions (which are not provided in the object base).

- All renormalization procedures can be carried out treating the data as if it were dimensional with a consistent set of units.

- Any application code that internally uses consistent normalization can use the data provided in an object base without modification or transformation to the code's internal normalization.

EXAMPLE 4 A CFD application code that internally uses inconsistent normalization could easily read and write data to a nondimensional object base that conforms to the above stipulations. On output, the code could renormalize data so it is consistently normalized. Probably, the easiest method would be to remove the molecular viscosity scale (μ_∞), and only use L, ρ_∞ and c_∞ for all normalizations (recall these are dimensional scales). The only change from the above example would be the nondimensionalization of viscosity, which would become, $\tilde{\tilde{\mu}} = \mu/(\rho_\infty c_\infty L)$. The code could then output all field data as,

$$\begin{array}{lll} \tilde{x}_{ijk} = x_{ijk}/L, & \tilde{u}_{ijk} = u_{ijk}/c_\infty, & \tilde{\rho}_{ijk} = \rho_{ijk}/\rho_\infty, \\ \tilde{y}_{ijk} = y_{ijk}/L, & \tilde{v}_{ijk} = v_{ijk}/c_\infty, & \tilde{p}_{ijk} = p_{ijk}/(\rho_\infty c_\infty^2), \\ \tilde{z}_{ijk} = z_{ijk}/L, & \tilde{w}_{ijk} = w_{ijk}/c_\infty, & \tilde{\tilde{\mu}}_{ijk} = \mu_{ijk}/(\rho_\infty c_\infty L), \end{array}$$

and output the freestream reference quantities,

$$\begin{array}{ll} \tilde{u}_\infty = u_\infty/c_\infty, & \tilde{\rho}_\infty = \rho_\infty/\rho_\infty = 1, \\ \tilde{v}_\infty = v_\infty/c_\infty, & \tilde{p}_\infty = p_\infty/(\rho_\infty c_\infty^2) = 1/\gamma, \\ \tilde{w}_\infty = w_\infty/c_\infty, & \tilde{\tilde{\mu}}_\infty = \mu_\infty/(\rho_\infty c_\infty L) \sim O(1/Re), \end{array}$$

where γ is the specific heat ratio (assumes a perfect gas) and Re is the Reynolds number.

On input, the flow solver should be able to recover its internal normalizations from the data in a nondimensional object base by treating the data as if it were dimensional.

— **NondimensionalParameter:** When **class** = **Nondimensional_parameter**, the data is a nondimensional parameter (or array of nondimensional parameters). Examples include Mach number, Reynolds number and pressure coefficient. These parameters are prevalent in CFD, although their definitions tend to vary between different application codes.

Nondimensional parameters are distinguished from other data classes by the fact that they are *always* dimensionless. In a completely nondimensional object base, they are distinct in that their normalization is not necessarily consistent with other data.

Typically, the **units**, **exponents** and **conversion** qualifiers are not used for nondimensional parameters; although, there are a few situations where they may be used (these are discussed below). Rather than rely on optional qualifiers to describe the normalization, any nondimensional parameter shall be accompanied by their defining scales;

EXAMPLE 5 An example is Reynolds number defined as $Re = VL/\nu$, where V, L and ν are velocity, length and viscosity scales, respectively. Note that these defining scales may be dimensional or nondimensional data.

In certain situations, it may be more convenient to use the optional qualifiers of **data_array** to describe the normalization used in nondimensional parameters. These situations must satisfy two requirements: First, the defining scales are dimensional; and second, the nondimensional parameter is a normalization of a single 'raw' data quantity and it is clear what this raw data is.

EXAMPLE 6 Examples that satisfy this second constraint are pressure coefficient, where the raw data is static pressure, and lift coefficient, where the raw data is the lift force. Conversely, Reynolds number is a parameter that violates the second requirement — there are three pieces of raw data rather than one that make up Re.

For nondimensional parameters that satisfy these two requirements, the qualifiers **units**, **exponents** and **conversion** may be used to recover the raw dimensional data.

— **dimensionless_constant:** When **class** = **dimensionless_constant**, the data is a constant (or array of constants) with no associated dimensional units. The **units**, **exponents** and **conversion** qualifiers are not used.

EXPRESS specification:

```
*)
ENTITY data_array;
  descriptions : LIST OF text;
  id           : label;
  dimension    : INTEGER;
  counts       : ARRAY [1:dimension] OF INTEGER;
  data         : explicit_table_function;  -- Part 50
  classifier   : OPTIONAL data_name;
  units_class : OPTIONAL data_class;
WHERE
  wr1 : data.shape = total_number_of_elements(counts);
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_data_array FOR data_array;
  ONEOF(dimensional_data_array,
        nondimensional_data_array);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

descriptions: is annotation;

id: is a user-specified instance identifier;

dimension: The number of dimensions in the multidimensional data array;

counts: The array sizes for each dimension;

data: The data values;

classifier: An identifier or name that identifies and describes the quantity being stored;

units_class: The class of data;

Formal propositions:

wr1: The **data** shall have the same number of elements as required by **counts**.

5.4.6 dimensional_data_array

A **dimensional_data_array** is a **data_array** holding data that is dimensional.

EXPRESS specification:

```
*)
ENTITY dimensional_data_array
  SUBTYPE OF (data_array);
  units       : OPTIONAL dimensional_units;
  exponents   : OPTIONAL dimensional_exponents; -- Part 41 measure_schema
  conversion  : OPTIONAL data_conversion;
WHERE
  wr1 : NOT EXISTS(SELF/data_array.units_class);
END_ENTITY;
(*
```

Attribute definitions:

units: The dimensional units of the data;

exponents: The dimensional exponents;

conversion: The normalization.

Formal propositions:

wr1: The (inherited) **units_class** attribute shall have no value.

5.4.7 nondimensional_data_array

A **nondimensional_data_array** is a **data_array** holding data that is not dimensional.

EXPRESS specification:

```
*)
ENTITY nondimensional_data_array
  SUBTYPE OF (data_array);
WHERE
  wr1 : (units_class <> dimensional) AND
        (units_class <> unspecified);
END_ENTITY;
(*
```

Formal propositions:

wr1: The value of the (inherited) **units_class** attribute shall be neither **dimensional** nor **unspecified**.

5.5 data_array_schema function definitions

5.5.1 total_number_of_elements

total_number_of_elements takes an aggregate of positive integers as an argument and returns the product of the values.

EXAMPLE 1 If the argument represents the dimensions of a multi-dimensional array then the calculated result is the number of elements necessary for an equivalent single-dimensional array.

EXPRESS specification:

```
*)
FUNCTION total_number_of_elements (arg : AGGREGATE OF INTEGER) : INTEGER;
  LOCAL
    total : INTEGER := 1;
  END_LOCAL;
  REPEAT i := 1 TO SIZEOF(arg);
    total := total*arg[i];
  END_REPEAT;
  RETURN(total);
END_FUNCTION;
(*
```


Argument definitions:

arg: An aggregate of integers.

RETURNS: The product of the elements of **arg**.

EXPRESS specification:

```
*)
END_SCHEMA; -- end of data_array_schema
(*
```


Annex A
(normative)
Short names of entities

Table A.1 provides the short names of entities specified in this part of ISO 10303. Requirements on the use of short names are found in the implementation methods included in ISO 10303.

NOTE The short names are available from the Internet — see annex C.

Annex B
(normative)
Information object registration

B.1 Document identification

To provide for unambiguous identification of an information object in an open system, the object identifier

{ iso standard 10303 part(5s) version(-1) }

is assigned to this part of ISO 10303. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

B.2 Schema identification

To provide for unambiguous identification of the **mesh_topology_schema** in an open information system, the object identifier

{ iso standard 10303 part(5s) version(1) object(1) mesh-topology-schema(1) }

is assigned to the **mesh_topology_schema** schema (see 4). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

To provide for unambiguous identification of the **data_array_schema** in an open information system, the object identifier

{ iso standard 10303 part(5s) version(1) object(1) data-array-schema(1) }

is assigned to the **data_array_schema** schema (see 5). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

Annex C
(informative)
EXPRESS listing

This annex references a listing of the EXPRESS entity names and corresponding short names as specified in this part of ISO 10303. It also references a listing of each EXPRESS schema specified in this part of ISO 10303, without comments or other explanatory text. These listings are available in computer-interpretable form and can be found at the following URLs:

Short names: <http://www.mel.nist.gov/div826/subject/apde/snr/>
EXPRESS: <http://www.mel.nist.gov/step/parts/part5w/cd/>

If there is difficulty accessing these sites contact ISO Central Secretariat or contact the ISO TC 184/SC4 Secretariat directly at: sc4sec@cme.nist.gov.

NOTE The information provided in computer-interpretable form at the above URLs is informative. The information that is contained in the body of this part of ISO 10303 is normative.

Annex D
(informative)
EXPRESS-G diagrams

The diagrams in this annex correspond to the EXPRESS schemas specified in this part of ISO 10303. The diagrams use the EXPRESS-G graphical notation for the EXPRESS language. EXPRESS-G is defined in annex D of ISO 10303-11.

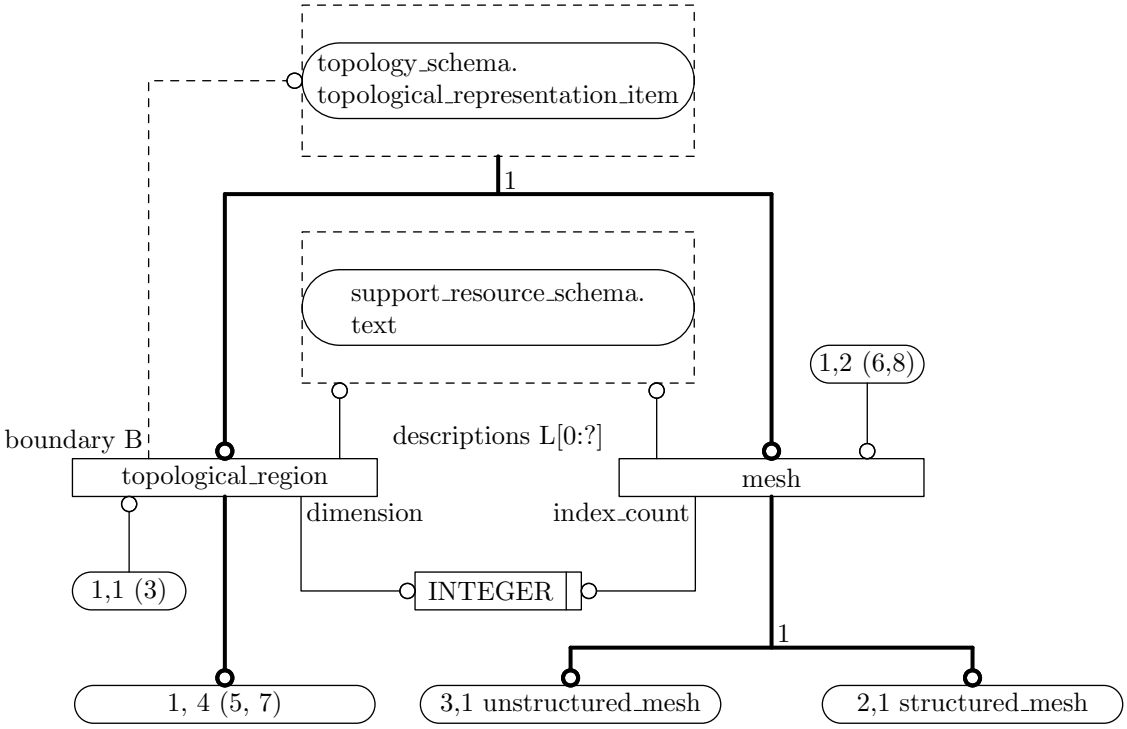


Figure D.1 – Entity level diagram of mesh_topology_schema schema (page 1 of 8)

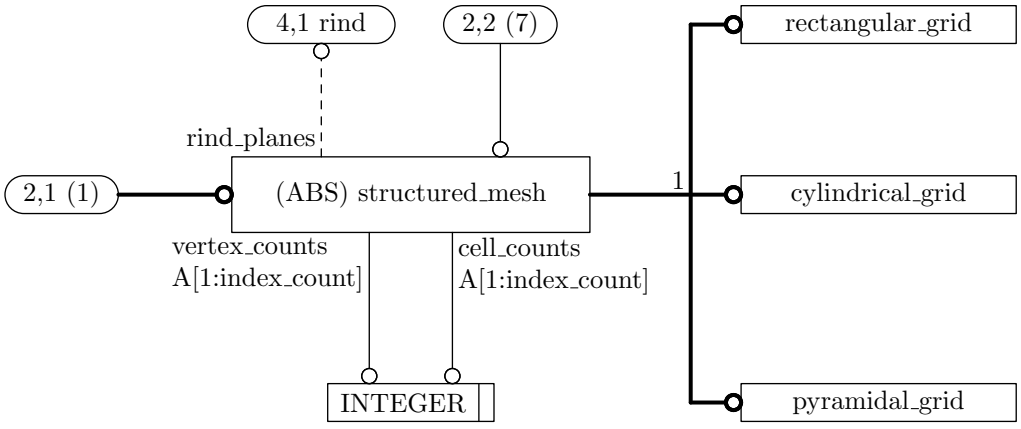


Figure D.2 – Entity level diagram of mesh_topology_schema schema (page 2 of 8)

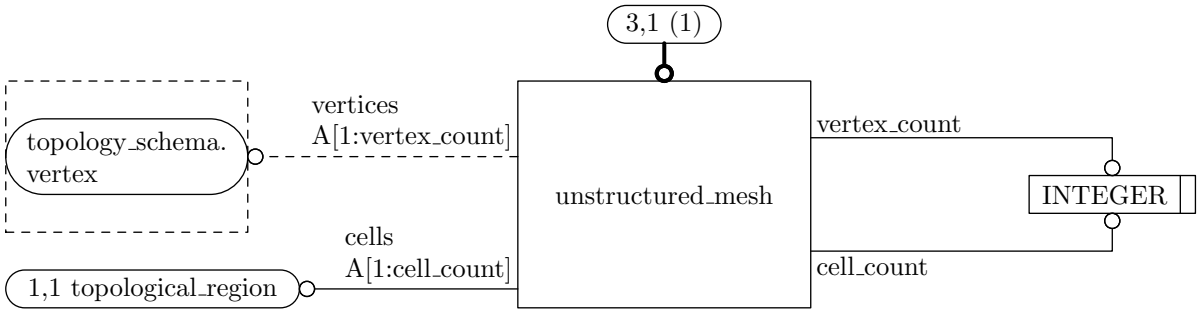


Figure D.3 – Entity level diagram of mesh_topology_schema schema (page 3 of 8)

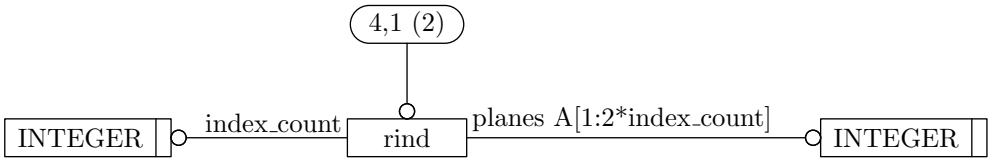


Figure D.4 – Entity level diagram of mesh_topology_schema schema (page 4 of 8)

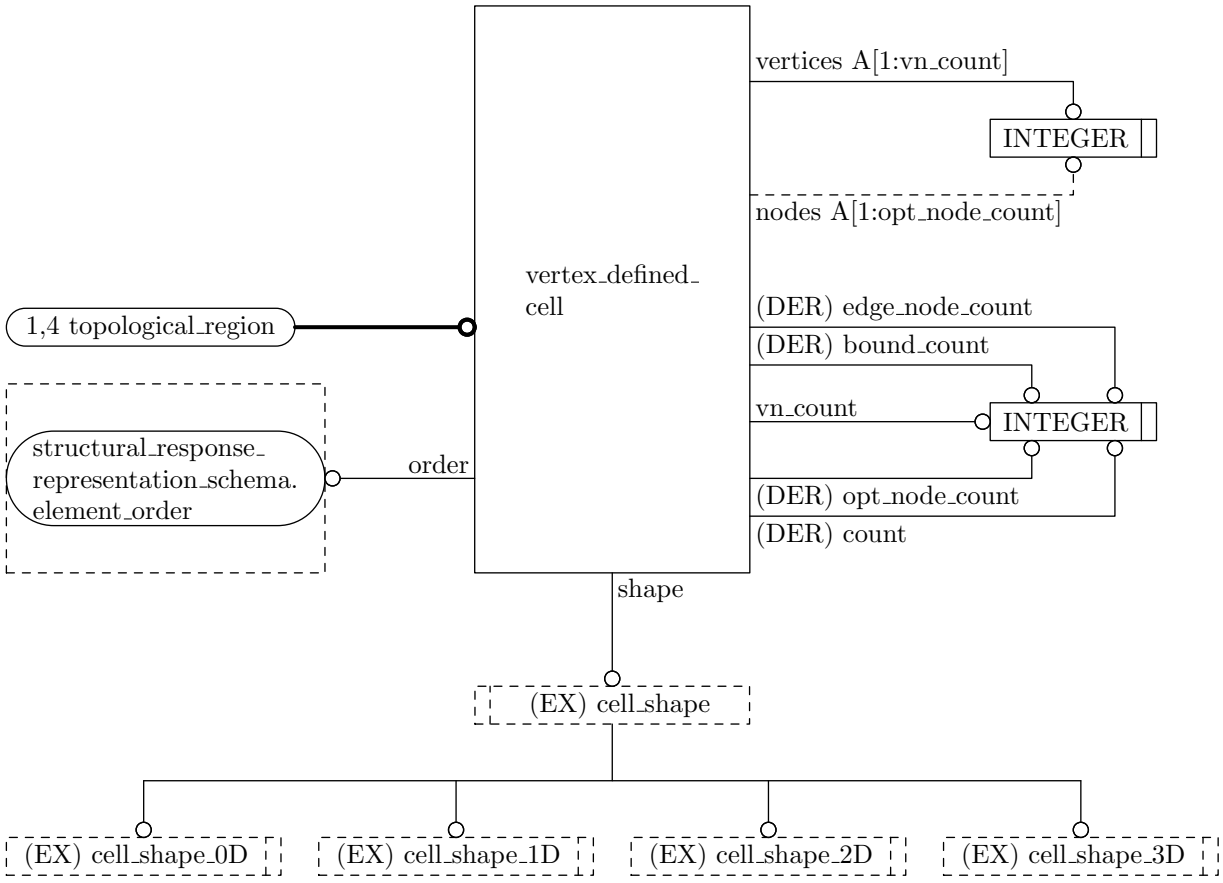


Figure D.5 – Entity level diagram of mesh_topology_schema schema (page 5 of 8)

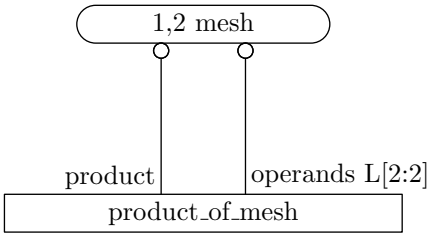


Figure D.6 – Entity level diagram of mesh_topology_schema schema (page 6 of 8)

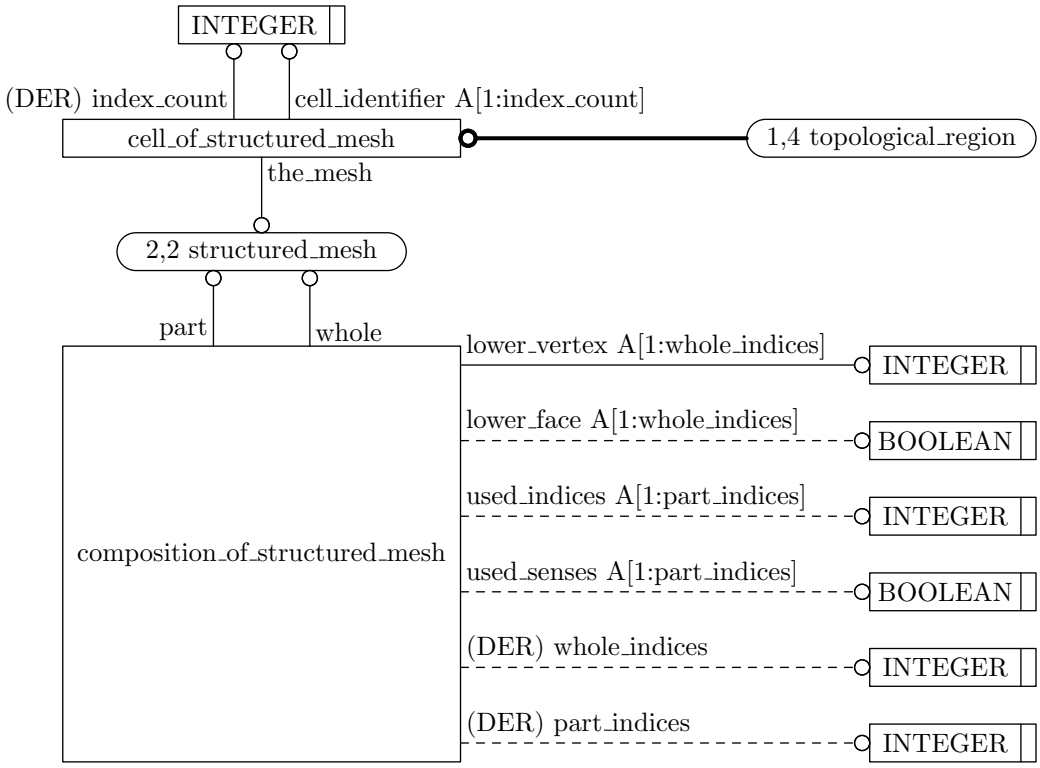


Figure D.7 – Entity level diagram of mesh_topology_schema schema (page 7 of 8)

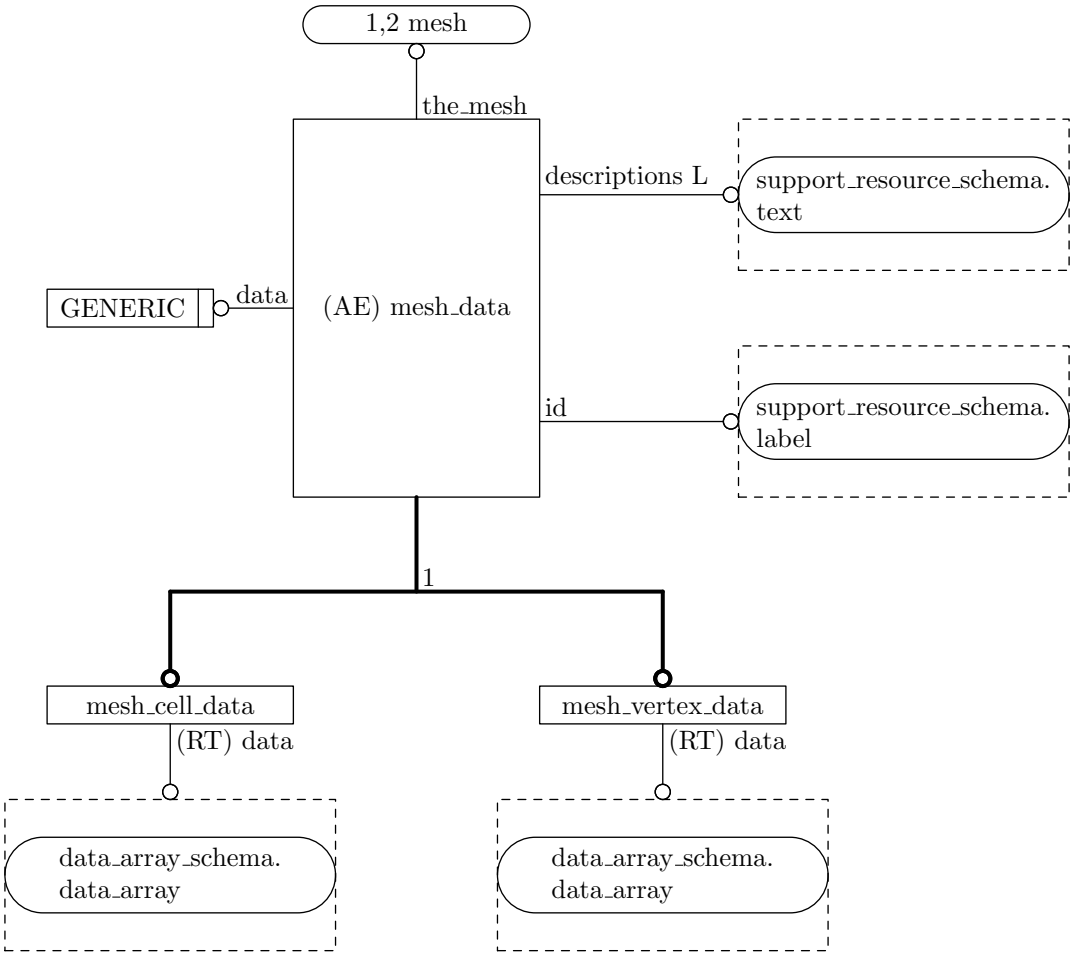


Figure D.8 – Entity level diagram of mesh_topology_schema schema (page 8 of 8)

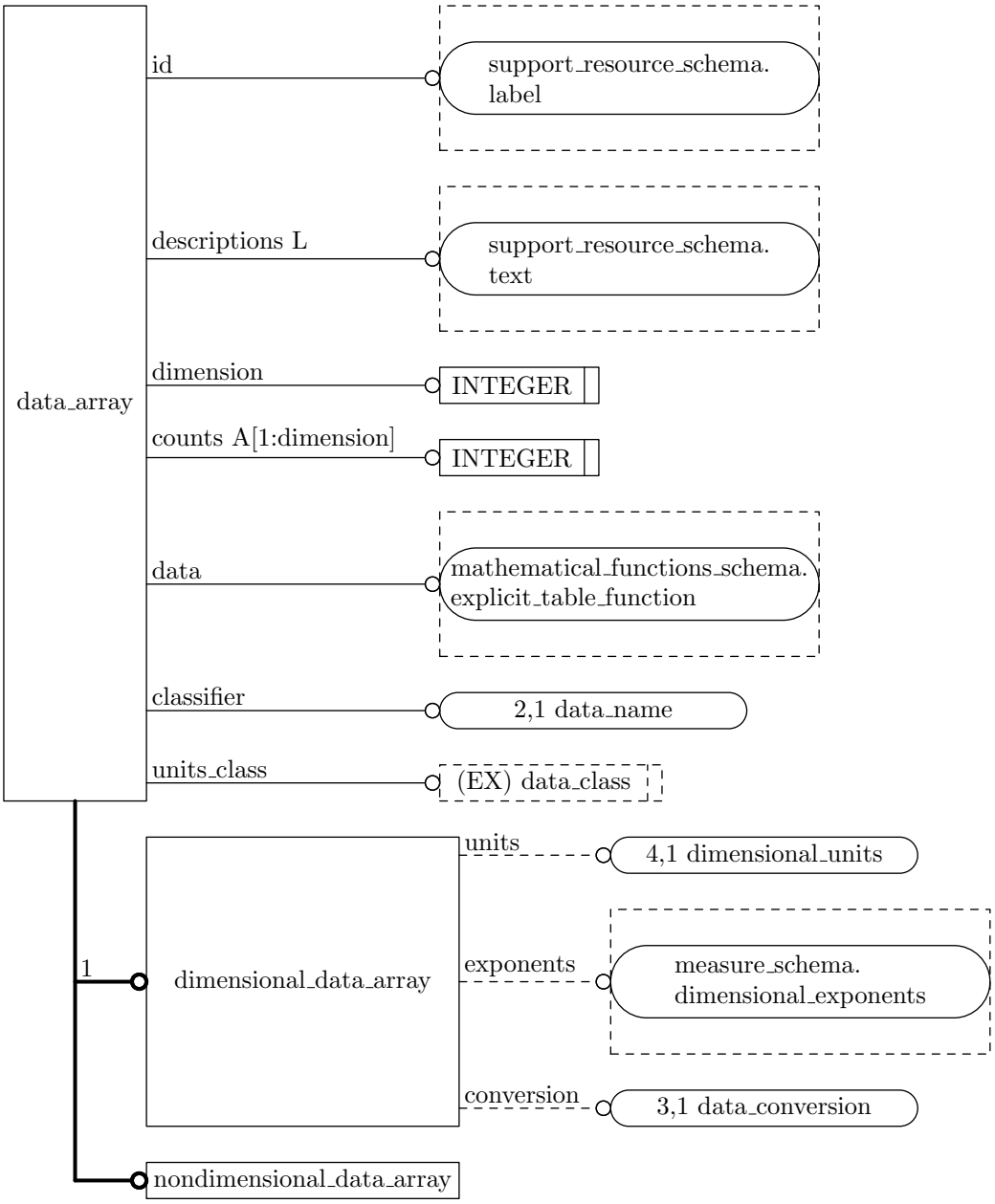


Figure D.9 – Entity level diagram of data_array_schema schema (page 1 of 4)

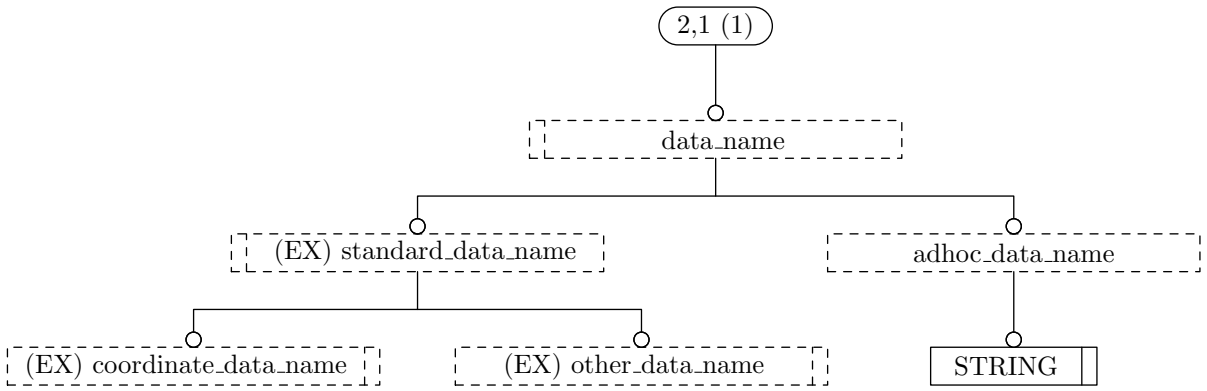


Figure D.10 – Entity level diagram of data_array_schema schema (page 2 of 4)

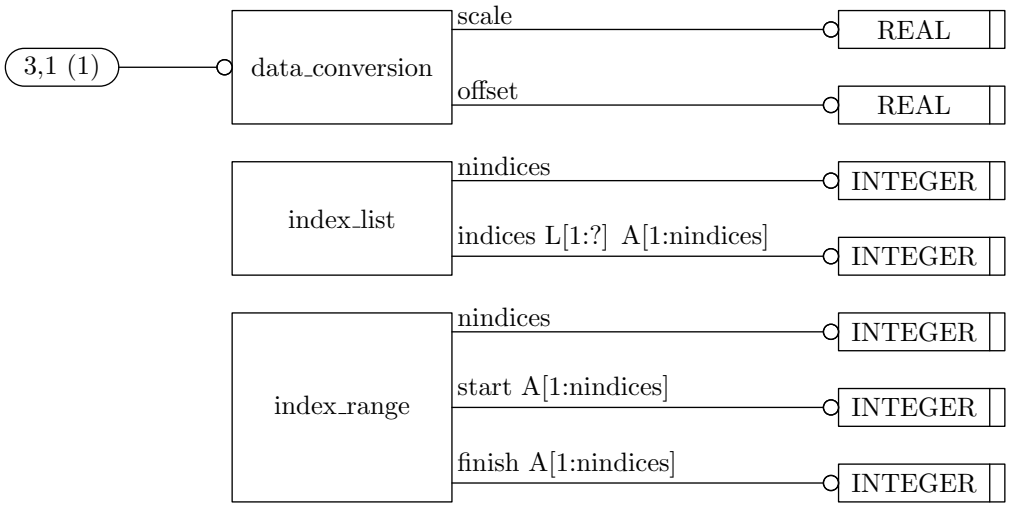


Figure D.11 – Entity level diagram of data_array_schema schema (page 3 of 4)

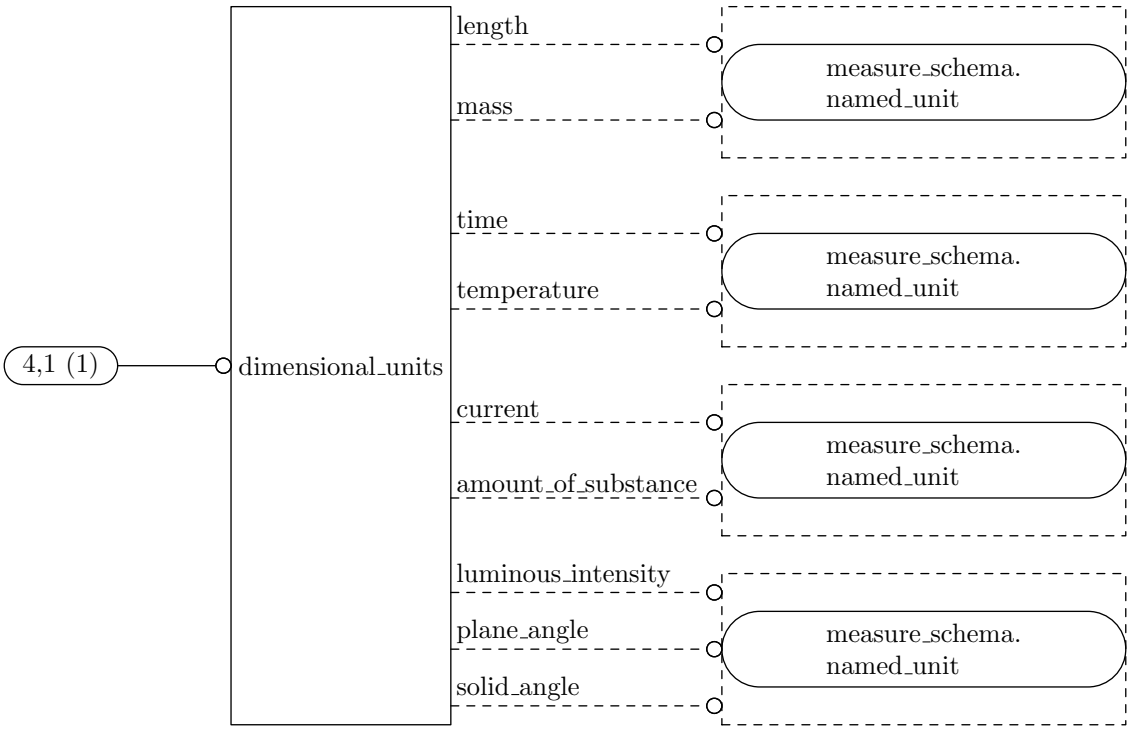


Figure D.12 – Entity level diagram of data_array_schema schema (page 4 of 4)

Index

